Independent Comparison of Popular DPI Tools for Traffic Classification

Adaptació de Volunteer-Based System per Mac OS X

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Albert Trelis Saiz

Director: Pere Barlet

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Projecte Final de Grau

Octubre 2016

Resum

L'any 2014 uns doctorants de l'UPC van modificar un software anomenat *Volunteer-Based System (VBS)*[1] perquè no només capturés paquets i els agrupés en traces, sinó perquè les traces guardessin informació sobre el procés que l'havia generat. Usant el *VBS* van crear una gran base de dades amb tràfic que havien generat d'aplicacions específiques per a poder realitzar un estudi comparatiu de les diverses *eines DPI* (*Deep Package Inspection*)[2] que hi havia al mercat. Es van analitzar un total de 7 eines i els resultats obtinguts van tenir una gran repercusió dins del món de l'anàlisi de paquets [3].

La finalitat d'aquest projecte de final de grau és ajudar a millorar i ampliar aquest software perquè pugui capturar paquets de més aplicacions i així poder realitzar estudis més amplis i acurats. L'objectiu principal és aconseguir que el software VBS funcioni no només en sistemes Windows i Linux, sinó també en sistemes Mac OS X ja que en els últims anys hi ha hagut un increment dels ordinadors i dispositius que usen aquest sistema operatiu.

Índex

Índex de figures

Índex de taules

Índex de codis

1. Introducció

1.1 Formulació del problema

Des del 2007 amb la sortida dels primers smartphones, les tecnologies s'han integrat cada vegada més al dia a dia de les persones. Actualment, la majoria de la població mundial disposa d'ordinadors, ja siguin de sobretaula o portàtils, i de smartphones o tablets que usen en la seva vida quotidiana. A més a més, ja es comencen a veure projectes relacionats amb Internet of Things, com per exemple neveres amb connexió a Internet, i projectes de Smarts Cities, concepte que consisteix en repartir diversos dispositius arreu d'una ciutat que, connectats a un servidor comú, envien dades en temps real.

Tot i estar rodejats de tecnologia, la majoria de les persones no són conscients de com funciona. Desconeixen la quantitat d'informació que es transporta a través de la xarxa amb un simple clic a l'hora d'obrir una aplicació o una pàgina web des d'un dispositiu connectat a Internet.

A causa d'això, en els últims anys, el nombre d'organitzacions criminals ha crescut de manera desmesurada. La majoria tenen com a objectiu interceptar qualsevol tipus d'informació i usar-la per a benefici propi. Això ha provocat que les grans companyies s'hagin centrat en intentar mantenir no només la seva empresa sergura, sinó també els seus clients. Una gran part del pressupost del que disposen es destina a la protecció de dades i informació confidencial per evitar que malfactors puguin obtenir quelcom que els pugui perjudicar.

Hi ha infinitat d'estratègies diferents per fer més segura una empresa. Les més connegudes són divulgar entre els treballadors i clients una certa metedologia de treball per evitar que siguin ells mateixos els qui facilitin informació confidencial a persones no desitjades. També s'han centrat en realitzar diversos anàlisis de virus i vulnerabilitats periódicament per evitar infeccions i atacs.

No obstant això, hi ha hagut companyies que han anat més enllà i s'han especialitzat en capturar el tràfic que passa a través d'un punt de la xarxa i analitzar-ne els paquets amb l'objecitu de detectar software maliciós. D'aquesta manera poden bloquejar-los abans

de que arribin als ordinadors interns. Aquestes tècniques usen software *DPI*[2]. La seva funcionalitat principal és capturar el tràfic que pasa per un punt de la xarxa i, paquet a paquet, analitzar el header, les adresses IP i ports, i el payload[4] tot buscant IPs perilloses o patrons que puguin tenir un contingut que apunti que aquest és de caràcter maliciós.

1.2 Contextualització

1.2.1 Definició del context

Amb els anys, la tècnica de *Deep Package Inspection* ha anat evolucionant. Són moltes les empreses que competeixen entre elles per oferir aquest tipus de software als seus clients, els quals l'únic que busquen és controlar el tràfic que passa per la seva xarxa per incrementar la seguretat.

No ha sigut fins al 2011 que s'ha disposat d'un software que permet avaluar l'eficiència i precissió dels programes DPI. *Volunteer-Based Software* (VBS) [1] és capaç d'analitzar tot el tràfic de la xarxa, crear-ne traces i etiquetar-les amb el nom de l'aplicació que les ha generat. D'aquesta manera es pot obtenir un informe amb els percentatges del tràfic capturat i es pot realitzar un estudi comparant els resultats de les diverses eines DPI del mercat.

Particularment, aquest TFG s'ha centrat en l'ampliació del software *VBS* per aconseguir que funcioni en sistemes *Mac OS X*. D'aquesta manera, en un futur, es podrà ampliar el nombre d'aplicacions que s'inclouen a l'estudi per comparar les eines *DPI*.

1.2.2 Actors implicats

A continuació, es descriu el rol de cada una de les parts involucrades en el desenvolupament del projecte i els potencials destinataris que es beneficiaran del resultat d'aquest treball.

Desenvolupador: Aquesta és la persona més activa del projecte. És el responsable de recopilar, estructurar i redactar l'informació. A més a més, també desenvoluparà la part tècnica.

Director: Pere Barlet-Ros, professor agregat del departament d'Arquitectura de Computadors a l'Universitat Politècnica de Catalunya, és l'encarregat de guiar i assistir en aquest projecte.

Usuaris Beneficiats: Principalment, el beneficiaris d'aquest projecte són les empreses que desenvolupen software DPI, ja que podran ampliar la quantitat d'aplicacions que es tenen en compte a l'estudi comparatiu i obtenir resultats més extensos i acurats.

1.3 Estat de l'Art

Mac OS X és un sistema operatiu per ordinadors que en els últims anys ha guanyat molta quota de mercat davant dels seus competidors directes, *Windows* i *Linux*. Això ha provocat que qualsevol empresa o persona que vulgui desenvolupar un software, es vegi obligat a adaptar-lo perquè funcioni en aquest sistema operatiu si vol que el projecte tinguin el major èxit possible.

El software *Volunteer-Based System (VBS)* s'ha vist també afectat. Si en un futur es volgués realitzar un estudi comparatiu de les diferents eines d'anàlisi de paquets, és completament imprescindible que l'estudi inclogui aplicacions que s'usen en sistemes Mac OS X, ja que un alt percentatge del tràfic que circula per la xarxa correspon a aquest sistema operatiu.

En el següent gràfic es pot observar la quota de mercat de cada sistema operatiu. Els tres més importants i que pot analitzar el software *VBS* són: Windows, Linux i Mac OS X amb unes quotes de mercat del 57%, 16% i 14% respectivament [5].

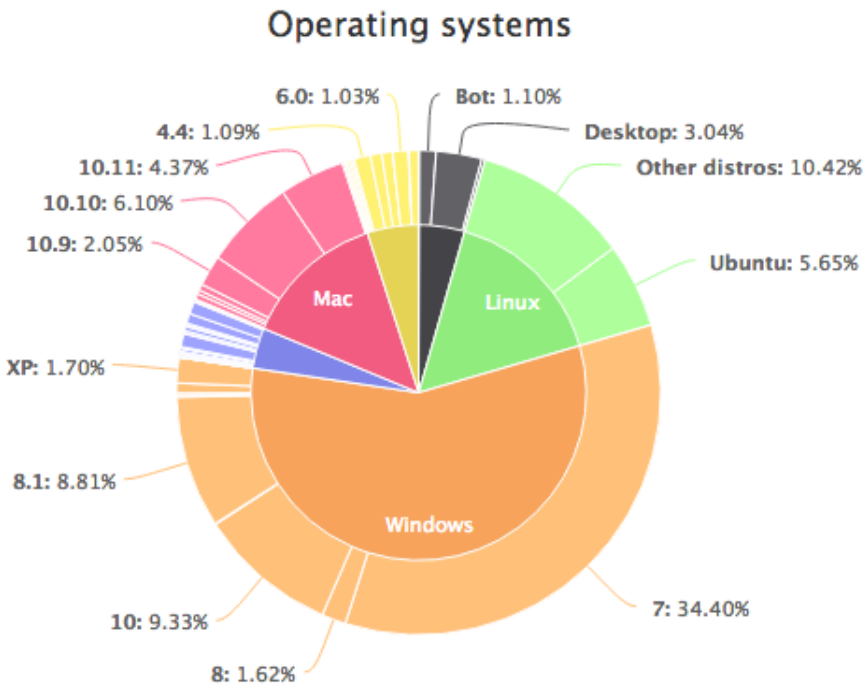


Figura 1 Quota de mercat sistemes operatius

2. Abast del projecte

2.1 Definició d'abast

Aquest projecte s'orienta en la modificació del *software VBS*, que funciona en sistemes Linux i Windows, per a què també funcioni en el sistema operatiu Mac OS X. No obstant això, l'adaptació d'aquest software no es podrà dur a terme si no es tenen consolidats un conjunt de coneixements sobre els diversos sistemes operatius i les seves llibreries, l'estructura del VBS i el funcionament de la captura i recepció de paquets. És per això, que amb la finalitat d'aconseguir una base teòrica sòlida, s'ha otorgat un pes considerable a la part bibliogràfica d'aquest àmbit.

En primer lloc, s'explica a nivell conceptual quines llibreries s'han utiltizat i per a què serveixen, la tècnica que s'ha usat per intentar crear una llibreria que funcioni en *Mac OS X* i es comenta en profunditat l'estructura i funcionament del software que s'ha d'adaptar, el *Volunteer-Based System*.

En segon lloc, es mostra la part de desenvolupament. Per una banda, s'expliquen els diversos errors obtinguts al llarg de l'adaptació del software i com solucionar-los i per l'altra, quines han sigut les modificacions i implementacions que s'han realitzat de les diverses classes.

Per últim, es fa un petit anàlisi comparatiu entre dues eines DPI de software lliure utilitzant una petita base de dades generada amb tràfic de diverses aplicacions de *Mac OS X*. A més a més, al final del projecte s'inclou un annex amb totes les comandes usades per a configurar l'entorn de treball: compilació de codi i estructura d'arxius entre d'altres.

2.2 Objectiu

L'objectiu d'aquest projecte és adaptar el software ja existent anomenat *Volunteer-Based System* perquè es pugui executar en ordinadors amb el sistema operatiu *Mac OS X*. D'aquesta manera es podran analitzar aplicacions que no existeixen en altres sistemes operatius.

2.3 Riscos i obstacles

Hi ha diversos factors potencials que poden interferir en les expectatives fixades en un començament. En aquest cas, els principals condicionants fan referència al temps, coneixements sobre la matèria i l'alta probabilitat d'incompatibilitats de llibreries i funcions entre els diferents sistemes operatius.

L'entrega d'aquest projecte està establerta en una data en concret i per tant, el període de temps que es pot invertir en el seu desenvolupament ha de respectar aquesta data. Això suposa que qualsevol contratemps que es produeixi podria afectar el projecte i conduir-lo a l'incompliment dels plaços. Ja que no es concep aquest escenari, la planificació temporal s'ha realitzat de tal manera que si hi hagués algun obstacle, aquest no afectés al flux del treball.

Un altre aspecte a considerar és el grau de coneixement previ de l'alumne sobre la materia a tractar. Això influirà parcialment en la velocitat de l'implementació del projecte. A causa de que el software a implementar és depenent de diverses llibreries i altres programes, pot provocar que sorgeixin diversos errors dels quals és molt complicat cercar informació a l'hora de realitzar el canvi de sistema operatiu. Per tant, aquest és un factor important que pot arribar a condicionar el projecte.

2.4 Metodologia i rigor

A aquesta secció s'exposa el mètode de treball escollit per a dur a terme el projecte, el conjut d'eines seleccionades per a realitzar el seguiment d'aquest i la metodologia usada per validar el compliment dels objectius prefixats.

2.4.1 Métodes de treball

Amb la finalitat d'establir una diciplina per al correcte desenvolupament del projecte, la metodologia àgil SCRUM [6] ha sigut l'escollida. Es tracta d'un model àgil de desenvolupament que té definides les conductes que s'han d'adoptar per a finalitzar el treball dins del termini fixat.

SCRUM té com a característica principal la constant interacció entre el client i el desenvolupador, director i alumne en el nostre cas, per a realitzar un seguiment de l'estat del projecte. Aquest fet aporta flexibilitat davant dels canvis que es puguin arribar a dur a terme, ja que no estan vistos com a un problema, sinó com a quelcom necessari per a obtenir un millor resultat final.

Totes aquestes característiques fan que SCRUM s'ajusti a les necessitats requerides en aquest projecte ja que ha sigut clau el feedback no només del director, sinó també dels dos doctorats que van desenvolupar aquest software uns anys enrere.

2.4.2 Eines de seguiment

Durant el desenvolupament del projecte s'han usat diverses eines per assegurar el correcte procediment i garantitzar que l'integritat del treball està protegida davant de qualsevol circumstància no desitjada.

La principal eina de treball ha sigut el software anomenat Sublime Text Editor, editor de text que s'ha usat per a modificar les classes del software. S'ha optat per usar aquest senzill programa i no un de més complex com podria ser un entorn adaptat pel desenvolupament d'aplicacions perquè el programador està adaptat i acostumat a usar-los. A més a més, per a gestionar la feina feta dia a dia i mantenir-ne un control, s'apuntaven totes les tasques i progrés en un document que es revisava a diari.

Per altra banda, Dropbox ha sigut una eina clau per a guardar tots els fitxers automàticament. A més, s'ha usat aquest programa per a compilar i actualitzar a temps real els arxius tant a l'ordinador client com al servidor. D'aquesta manera, quan es modificava una classe, es podia disposar d'ella per a realitzar proves immediatament a totes les màquines.

Per evitar la pèrdua d'informació, Dropbox permet disposar en tot moment de com a mínim 4 còpies físiques simultànies completament actualtizades i una de virtual.

2.4.3 Mètodes de validació

El projecte ha estat avaluat amb certa periodicitat, a convenir amb les preferències del director, per assegurar un desenvolupament continuat i correcte del projecte. La comunicació entre el director i l'alumne s'ha realitzat a través de correu electrònic. En aquest, l'alumne explicava i detallava quin era el progrès realitzat i en quina etapa es trobava. A més a més, al final de cada tasca es realitzaven reunions per a analitzar la part enllestida i definir la següent.

3. Plec de condicions

3.1 Descripció i motivació

El projecte forma part del *software VBS* el qual requereix de com a mínim una màquina que faci de client i una de servidor.

En termes generals, el client és l'encarregat de capturar el tràfic que prové d'una interfície de xarxa concreta i analitzar-ne els paquets. Depenent de quin sigui el seu contigut (header i payload) determina com serà el paquet que s'afegirà a la traça. Simultàniament, crea una traça per a cada conjunt de paquets amb la mateixa IP origen i la mateixa IP destí. A més a més, analitza totes les aplicacions obertes, determina a quina correspon la traça i n'assigna el nom del procés. Un cop els paquests estan agrupats, es guarden a la base de dades local. Cada vegada que el client s'inicia, crea una còpia de la base de dades modificant el nom i l'extensió (.fdb) i la buida per començar des de zero. Aquest fitxer que es crea és enviat al servidor.

Mentre que el client és l'agent actiu, el servidor es manté a l'espera. Inicialitza totes les seves funcionalitats i es queda escoltat a un port determinat fins que el client crea i envia els arxius .fdb. El servidor els rep, els analitza i els guarda a una base de dades *MySQL* [7] per a la posterior generació dels arxius *PCAP* [8].

Aquest procés permet la creació de tràfic de xarxa específic guardat en arxius anomenats *PCAP* que contenen diverses traces etiquetades amb el nom de l'aplicació que ha generat els paquets. Sabent quines traces hi ha a cada arxiu *PCAP* es poden usar perquè les eines *DPI* els analitzin i comparar així els resultats obtinguts amb l'informació de la que disposem de cada traça per a poder determinar quina és la precisió de l'eina analitzada.

3.2 Entorn

El projecte pertany a la modalitat A, en la qual un alumne realitza un projecte proposat per un professor d'un departament de la Facultat d'Informàtica de Barcelona (FIB). En aquest

cas, el projecte va ser proposat pel professor Pere Barlet, del departament d'Arquitectura de Computadors.

3.3 Estat actual

Aquest projecte forma part d'un projecte de doctorat anomenat "Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification" desenvolupat per Tomasz Bujlow, Valentin Carela-Español i Pere Barlet (director) l'any 2014 a l'Universitat Politècnica de Catalunya.

Aquest projecte va ser pioner en el seu sector ja que no existia cap manera per a poder analitzar la precisió de les diferents eines DPI existents al mercat. Abans del 2014, eren les pròpies empreses qui enunciaven els percentatges de precisió del seu producte, cosa que provocava en el client un cert estat de desconfiança a causa de la poca objectivitat de les dades.

A l'estudi es van analitzar 6 eines: PACE, OpenDPI, L7-filter, nDPI, Libprotoident, i NBAR usant tràfic generat per aplicacions de sistemes operatius Linux (Ubuntu) i Windows (Windows XP i Windows 7).

A causa de l'èxit i la bona acceptació que va tenir en el sector, es va decidir ampliar-lo per a poder abarcar els canvis constants en el món de l'informàtica.

3.4 Disseny arquitectònic i implementació

L'arquitectura del projecte són un conjunt de classes Java i llibreries Dylib i Jnilib, que són les que conformen el client i servidor. El primer funcionarà en un sistema operatiu *Mac OS X* i el segon en un sistema *Linux* (Ubuntu).

La principal eina de desenvolupament per a dur a terme les tasques de programació de les classes és *Sublime Text Editor*. S'ha optat usar aquest senzill programa i compilar amb comandes des de la consola a causa de la complexitat de les dependències de les classes.

A nivell hardware, es disposa d'un ordinador de sobretaula que executa dues màquines virtuals: una amb *Ubuntu*, el servidor, i l'altra amb *Mac OS X El Capitan*, el client. A més a més, també es disposa d'un *MacBook Air* amb el mateix sistema operatiu que el client per a poder treballar a distància usant *TeamViewer* per a conectar-se amb l'ordinador de sobretaula.

3.5 Gestió de riscos

Els riscos associats a les eines i tecnologies utilitzades pel desenvolupament del projecte no suposen una gran amenaça per a l'evolució d'aquest.

Al tractar-se d'una aplicació Java, totes les llibreries usades tenen una llicència oberta i el tràfic utiltizat per a realitzar proves ha sigut creat pel propi desenvolupador i per tant, no hi ha cap incompatibilitat amb la llei de protecció de dades.

El punt fort de *Java* és que mitiga gran part de les complicacions ja que disposa d'una extensa documentació proporcionada des de fonts oficials. També es pot trobar molta ajuda a comunitats tecnològiques. No obstant això, en aquest cas, al ser un projecte únic hi ha molts aspectes dels que no hi ha documentació.

3.6 Competències tècniques de l'especialitat

Aquest projecte pertany a l'especialitat de Tecnologies de l'Informació i les competències tècniques escollides en el moment de l'inscripció del TFG van ser les següents:

- **CTI1.3**: Seleccionar, desplegar, integrar i gestionar sistemes d'informació que satisfacin les necessitats de l'organització amb els criteris de cost i qualitat identificats.
- **CTI2.2:** Administrar i mantenir aplicacions, sistemes informàtics i xarxes de computadors (els nivells de coneixement i de comprensió són a les competències tècniques comunes).
- **CTI2.3:** Demostrar comprensió, aplicar i gestionar la garantia i la seguretat dels sistemes informàtics (CEIC6).
- **CTI3.3:** Dissenyar, implantar i configurar xarxes i serveis.

La descripció del projecte donada en seccions anteriors posa en manifest que s'han abordat els espectes relacionats amb les competències CTI1.3, CTI2.2 i CTI3.3.

Per altra banda, la competència CTI2.3 no s'ha tractat en profunditat ja que no s'ha hagut de reimplementar cap sistema de seguretat.

4. Planificació temporal

4.1 Planificació estimada del projecte

El projecte té una duració aproximada de cinc mesos. En un principi es va decidir començar al mes març i acabar al mes de juny, però per diverses raons es va aplaçar. És per això, que la planificació que es mostra a continuació té en compte aquest aplaçament i per tant es considera que el projecte comença a l'1 de juny i acaba a finals d'octubre amb la presentació oral. Durant aquests cinc mesos, les hores de dedicació diàries flactuen en funció del calendari laboral i lectiu. No obstant això, el promig d'hores dedicades es pot estimar en 4.5 hores diàries. Suma un total d'unes 630 hores.

4.2 Descripció de les tasques

1. **Gestió del projecte:** són les tasques que es duen a terme a l'assignatura de Gestió de Projectes (GEP). A través d'aquestes tasques es defineix l'abast i context del projecte, es realitza l'estimació temporal de la duració d'aquest i es realitza un estudi sobre l'impacte econòmic i la sostenibilitat. S'ha decidit posar al diagrama que l'assignatura comença el dia 1 de juny ja que a causa de l'aplaçament del TFG ha sigut necessari refer tota la feina feta durant el quatrimestre de primavera.

2. **Adaptació a l'entorn:** Al ser l'ampliació d'un projecte de doctorat, ha sigut necessari estudiar i entendre cadascuna de les classes. En aquesta tasca ha sigut necessari invertir molt de temps, ja que per una banda s'ha hagut d'esbrinar el funcionament de tots els scripts i wrappers tant del client com el servidor, per cridar les classes i les comandes que s'executaven. Per altra banda, hi ha totes les classes que usen el client i el servidor per capturar els paquests, crear les traces, crear i guardar les bases de dades i transmetre els arxius. S'han d'entendre perfectament juntament amb la creació dels threads de cada una ja que són les classes que es reimplementaran.

3. **Configuració de l'entorn:** Per a la correcta execució del software, és necessari assignar una màquina que faci de client i una altra de servidor i configurar totes les màquines virtuals usades per a la virtualització dels sistemes operatius. A més a més, és necessari instal·lar totes les llibreries i programes necessaris per a la correcta execució del codi.

4. **Identificació i solució de problemes de compatibilitat:** Ja que l'objectiu és adaptar el software a *Mac OS X*, una de les primeres coses que s'ha de fer és identificar quines parts del codi s'han de canviar i, si és possible, solucionar el problema. A més a més, també es va investigar quines eren les llibreries necessàries per a la correcta execució del codi i si eren compatibles amb el sistema operatiu.

5. **Desenvolupament:** Aquest punt està dividit en dos subapartats ja que en una primera planificació del projecte es va escollir un camí per assolir l'objectiu però es va haver de canviar perquè es va arribar a la conclusió de que no era factible.

    - **Adaptació de la llibreria JnetPcap a *Mac OS X*:** La llibreria usada per a la captura de paquets s'anomena JnetPcap i és compatible amb *Ubuntu* i *Windows*. L'objectiu és veure on s'usa la llibreria en el client i identificar si és possible adaptar-la tota a *Mac OS X*, ja que encara no és compatible.
    - **Reimplementació amb la llibreria Pcap4j:** Una altra llibreria similiar a *Jnet-Pcap* és *Pcap4j*. Té gairebé les mateixes funcionalitats i té l'avantatge que és completament compatible amb el sistema operatiu *Mac OS X*. L'objectiu és reimplementar totes les classes que siguin necessàries del client perquè funcioni amb aquesta llibreria.

6. **Etapa final:** Per acabar, es fa una petita comparació entre dues *eines DPi* i es realitza una última revisió global del codi i la memòria. A més a més, es prepara la prensentació de la defensa oral que es du a terme a l'última setmana d'octubre.

4.3 Diagrama de Gantt

En aquest apartat es mostra una taula on s'especifica cada tasca i el temps necessari per dur-la a terme. Posteriorment es presenta gràficament en un diagrama de Gantt.

Tasca	Data Inici	Data Final	Dies
Gestió de projecte	**01/06/2016**	**20/06/2016**	**19**
Definició de l'abast i contextualització	01/06/2016	06/06/2016	5
Planificació temporal	06/06/2016	09/06/2016	3
Gestió econòmica i sostenibilitat	09/06/2016	12/06/2016	3
Revisió de les competències	12/06/2016	13/06/2016	1
Document final	13/06/2016	20/06/2016	7
Adaptació a l'entorn	**20/06/2016**	**01/07/2016**	**11**
Configuració de l'entorn	**01/07/2016**	**11/07/2016**	**10**
Identificació i solució de problemes de compatibilitat	**11/07/2016**	**26/07/2017**	**15**
Desenvolupament	**25/07/2016**	**03/10/2016**	**70**
JnetPcap	25/07/2016	29/08/2016	35
Pcap4j	29/08/2016	03/10/2016	35
Etapa final	**03/10/2016**	**24/10/2016**	**21**

Taula 1 Planificació del projecte

Figura 2 Diagrama de Gantt

4.4 Recursos

A continuació s'enumeren els recursos usats durant el projecte:

- **Hardware:**
  - Ordinador portàtil *MacBook Air* amb *Mac OS X El Capitan*.
  - Ordinador de sobretaula amb *Windows 7*.
- **Software:**
  - Editor de text *Sublime Text Editor*.
  - VMWare amb les màquines virtuals *Ubuntu 14.04* i *Mac OS X El Capitan*.
  - *Dropbox* per l'emmagatzematge del projecte i la sinconització entre les diferents màquines.
  - *Volunteer-Based System (VBS)*.

4.5 Valoració d'alternatives i pla d'acció

La planificació temporal mostrada anteriorment ha sigut dissenyada tenint en compte la càrrega lectiva i laboral que acompanya en paral·lel el projecte de final de grau. És per això que algunes tasques s'han solapat i algunes altres s'han allargat.

No obstant això, és necessari identificar els principals obstacles que podrien causar una desviació de la ruta fixada pel diagrama de *Gantt*. Aquests són el grau de desconeixament en certs espectes del sistema operatiu *Mac OS X* a nivell de software.

En conseqüència, s'ha fixat com a data límit el 15 d'octubre, uns dies abans de l'entrega final. Amb aquest marge de temps combinat amb dies lliures, serà suficient per solventar els imprevistos.

Aquests imprevistos estan vinculats amb la part pràctica de l'aplicació, sobretot a l'intentar adapatar la llibreria *JnetPcap* a *Msc OS X*. Si durant el desenvolupament del projecte es considera que existeix un perill de no complir amb el termini de finalització, es descartaran alguns objectius secundaris.

5. Fonaments Teòrics

5.1 Llibreries

En aquest apartat es defineixen quines són les principals llibreries usades en aquest projecte i quina és la seva funció.

5.1.1 SQLite

SQLite [9] és un sistema de gestió de bases de dades relacionals. A diferència dels sistemes de gestió client-servidor, el motor d'SQLite no és un procés independent amb el que el programa principal es comunica, sinó que és el propi programa qui crida la llibreria i utilitza les funcionalitats de SQLite a través de crides simples a subrutines i funcions.

Un avantatge d'usar SQLite davant de sistemes client-servidor és que són molt més eficients degut a que les crides a funcions tenen menys latència que la comunicació entre processos.A més a més, SQLite llegeix i escriu directament a un fitxer emmagatzemat a disk.

Per aquestes raons, la part del client usa aquest sitema en comptes del client-servidor, ja que és molt més eficient per emagatzemar totes les traces de manera temporal abans d'enviar-les al servidor. Per altra banda, el servidor si que usa MySQL ja que requereix molt més espai d'emmagatzematge per crear les traces finals PCAP i les dades no són temporals.

5.1.2 log4j

Log4j [10] és una llibreria adicional de java que permet al software mostrar missatges d'informació del que està passant, normalment conegut com a log. Té l'avantatge davant d'altres logs que és molt semblant a la comanda *System.out.println()*, això provoca que sigui molt més configurable permetent guardar tots els logs en un fitxer apart o mostrar-los per pantalla i que l'eficiència sigui molt elevada ja que no consumeix gaires recursos.

VBS usa aquesta llibreria per mantenir un seguiment de tots els threads durant l'execució del programa. Tots els logs es mostren per pantalla i es guarden a un fitxer de text.

5.1.3 Sigar

L'API de Sigar [11] proporciona una sèrie de mètodes que permeten obtenir informació del sistema operatiu, CPU [12], RAM [13] entre d'altres.

Aquesta llibreria s'usa molt poc en el projecte. L'única classe que l'utiltiza és la classe del handler que envia la següent informació al servidor: sistema operatiu i CPU.

5.1.4 jNetPcap

jNetPcap [14] és una llibreria open-source de java que conté un wrapper [15] per gaire bé totes les crides natives de la llibreria libpcap [16] (Linux) i WinPcap [17] (Windows), decodifica paquets capturats a temps real, té una gran llibreria de protocols d'Internet, permet als usuaris afegir protocols personalitzats usant l'SDK de Java [18] i utilitza una barreja d'implementacions natives i java per optimitzar la decodificació de paquets.

Aquesta llibreria és usada per les classes de Paquet Capturer per capturar el tràfic que genera el client i tractar cada paquet de manera individual depenent d'unes certes condicions. A més a més, també permet extreure tota l'informació que conté el paquet. És compatible amb una infinitat de protocols. Els usats en aquest projecte són: Ethernet [19], IPv4 [20], IPv6 [21], ICMP [22], HTTP [23], TCP [24] i UDP [25].

Un dels principals inconvenients que té aquesta llibreria, i és la raó principal del perquè s'ha hagut de precindir d'ella en aquest projecte, és perquè és compatible en sistems operatius Windows i Linux però no en Mac OS X.

5.1.5 Pcap4J

Pcap4J [26] és una llibreria usada per capturar, modificar i enviar paquets. Igual que *jNetPcap*, conté un wrapper que usa les llibreries natives *libpacap* i *WinPcap* per capturar paquests.

Les seves funcionalitats són capturar paquets a través de l'interfície de l'ordinador i convertir-los en objectes de Java. De cada un d'aquests es pot obtenir tota l'informació del paquet i també se'n poden crear de nous. També permet enviar paquets a través de la xarxa. Permet també crear protocols nous sense necessitat de modificar la llibreria de Pcap4J. És compatible amb tots els protocols usats en el projecte excepte HTTP.

S'ha escollit aquesta llibreria perquè els desenvolupadors van crear una nova versió compatible amb sistmes Mac OS X i té totes les funcionalitats necessàries per a la correcta captura de paquets.

5.2 Framework JNI

Java Native Interface (JNI) [27] és un framework de programació que permet que un programa escrit en Java i executat en la màquina virtual Java (JVM) pugui interactuar amb programes escrits en altres llenguatges de programació.

El JNI s'usa per escriure mètodes natius que permetin solventar situacions en les que una aplicació no pot ser escrita completament en Java. Les funcions natives s'implementen a arxius .c o .cpp per separat. Quan la màquina virtual invoca a la funció, li passa un punter *JNIEnv* i qualsevol número d'arguments declarats en el mètode Java. El punter JNIEnv *env és una estructura que conté l'interficie cap a la màquina virtual Java. Inclou totes les funcions necessàries per interactuar amb la JVM (Java Virtual Machine) i permet treballar amb els objectes Java. Resumint, amb JNI i usant JNIEnv es pot fer qualsevol cosa que el codi de Java pugui fer. Això si, amb una dificultat considerablement més alta.

En aquest projecte s'intenta usar per recomplilar tota la llibreria JnetPcap i crear-la en un format que entengui Mac OS X, el dylib.

5.3 Voluntary-Based System - VBS

5.3.1 Descripció

És un software que captura tràfic d'Internet i els agrupa en traces amb l'informació dels paquets i quina aplicació ha sigut la responsable de gernerar-los. El sistema consta de un o més clients instal·lats en diverses màquines i un servidor responsable de la recol·lecció de tota l'informació.

Ambdós, client i servidor, estan dissenyats per executar-se en segon pla i iniciar-se automàticament amb el sistema opearatiu. Per a que això sigui possible, s'usa *YAJSW* [28], un projecte open-source que permet la creació del servei.

5.3.2 Client

La tasca del client consisteix en registrar l'informació de cada paquet que passa a través d'una de les interficies de la màquina i categoritzar-los en diferents traces excepte els paquets de la xarxa local que són filtrats. Cada traça conté la següent informació: temps d'inici i final de la traça, nombre de paquets, les addreces IP local i remota, els ports local i remot, el protocol usat a la capa de transport i el nom de l'aplicació del client que ha creat els paquets de la traça.

El client també recol·lecta informació sobre tots els paquets associats amb cada traça: direcció de la comunicació, mida, TCP flags i el timestamp en relació amb l'anterior paquet. Aquesta informació es retransmet al servidor que ho emmagatzema tot per anàlisis futurs. El client consta de quatre mòduls que s'executen en threads separats: paquet capturer, socket monitor, flow generator i data transmiter.

Packet Capturer

Per capturar el tràfic s'usen les llibreries externes *Winpcap* (Windows) i *libpacap* (Linux i Mac OS X). A través de la llibreria *Pcap4J* que identifica quines interfícies seran les usades per a la captura de paquets i, la classe paquet capturer, crea un thread diferent per cada interfície escollida que s'executaran de manera independent.

Cada thread (classe capturer), s'encarrega de monitoritzar el tràfic de la xarxa, capturar-ne els paquets i guardar-los en una estructura local fins que el thread es tanca. Llavors, tots els paquets capturats s'envien a la classe packet capturer que els enviarà a socket monitor quan totes els threads de totes les interfícies es tanquin.

Socket Monitor

Tal i com diu el seu nom, la principal funcionalitat que té és controlar quins sockets estan oberts i obtenir-ne tota la informació necessaria a través de *Netstat* [29] a Linux, *Tcpviewcon* [30] a Windows i *lsof* [31] a Mac OS X, es monitoritzen tots els sockets TCP i UDP. Els sockets TCP inclouen informació sobre la destinació i l'origen ja que existeix una connexió. En canvi, als sockets UDP només hi ha informació sobre l'origen.

Flow Generator

Aquesta classe és una eina de monitorització que s'executa cada segon per assegurar-se que rep cada un dels flows. A més a més, a cada flow se li assigna el nom del socket i es guarda en una estructura local. Cada vegada que arriba un paquet, s'analitza i es guarda en un dels flows existents o se'n crea un de nou. Quan es tanca la connexió o s'esgota el timeout, que indica que ha passat massa temps sense actualitzar-se, el flow en qüestió es tanca i es guarda en una altra estructura que s'emmagatzemarà a una base de dades local del client.

En resum, s'encarrega de gestionar els flows. Agrupa els paquets i els flows, associa els flows amb el socket corresponent, tanca els flows si s'ha trobat el socket i els tanca en cas de time-out.

Database Handler

Una vegada s'han obtingut tots els flows i se'ls hi ha assignat el nom del socket al que corresponen, quan aquests es tanquen es guarden a la memòria fins que es crea una connexió amb SQLite i es podem emmagatzemar a un arxiu al client. Aquesta base de dades es crea des de nou sempre que s'executa el client, ja que es suposa que els resultats obtinguts en anteriors execucions ja s'han enviat al servidor.

Aquesta classe s'encarrega de realitzar les peticions d'incersió i update contra la base de dades local.

Data Transmiter

Abans de transmetre cap tipus de dades, el client s'autentifica amb el servidor i obté un identificador. Es realitzen dues connexions diferents, una per autentificar-se i l'altre per la transmissió de dades. Quan el nombre de flows és suficientment elevat, la base de dades SQLite es transmet al servidor i aquest la guarda a una base de dades permanent on també hi ha informació del client i el sistema operatiu usat.

5.3.3 Servidor

Vbs Server

El servidor es basa en tres threads. El primer thread autentifica els clients i els hi assigna un identificador únic a cada un d'ells. En canvi, el segon s'encarrega de rebre tots els arxius dels clients i guardar-los en una carpeta comuna on, periódicament, el tercer thread comprova que no hi hagi arxius corruptes i que el format de l'SQLite sigui el correcte. Si és així, s'insereixen a la base de dades MySQL [7] del servidor. Si les taules no existien a la base de dades del servidor, les crea. Per evitar que el tercer thread emmagatzemi un arxiu que encara no s'ha acabat de transmetre, s'usa un mètode de sincrontització que consisteix en modificar l'extensió del fitxer quan s'ha transmés correctament.

5.4 Eines DPI

En aquest apartat s'explicaran les dues eines que s'han utilitzat per a la petita comparativa duta a terme en aquest projecte.

5.4.1 nDPI

nDPI és una *eina DPI* open source que és uan versió optimitzada de *OpenDPI* [32]. A més a més de les funcionalitats que té *OpenDPI*, *nDPI* inclou més protocols (en soporta més de 100), suport per l'anàlisi de tràfic encriptat i la seva arquitectura és més escalable [33].

5.4.2 Libprotoident

Aquesta eina programada en C, té l'avantatge que és "lightweight". És a dir, examina només els primers quatre bytes de cada payload per direcció per minimitzar l'espai a disc necessari si es volen emmagatzemar les traces. A més a més, suporta més de 200 protocols diferents. En el nostre cas, a part d'aquesta eina també ha sigut necessaria fer ús d'un script que permeti llegir arxius PCAP [34][35].

6. Desenvolupament del projecte

Aquest projecte es pot dividir en quatre parts. La primera és tot el que comprén l'assignatura de Gestió de Projectes (GEP) explicat en apartats anteriors. La segona és un període d'anàlisis del software, on es busquen errors d'execució i es troben solucions. Un cop tot analitzat, s'escull un full de ruta que comprén la tercera i última part que consisteixen en l'intent d'adapcació de la llibreria jNetPcap a MAC OS X i reimplementar el software perquè funcioni amb la llibreria Pcap4j, respectivament.

6.1 Identificació d'errors

En aquesta fase es detecten, s'analitzen i es solucionen errors que surgeixen quan s'executa per primera vegada el software a una màquina amb sistema operatiu Mac OS X. Hi ha errors de tot tipus, alguns relacionats amb el wrapper, d'altres amb alguna classe del client o amb el daemon. La majoria d'aquest obstacles s'han pogut solucionar sense problemes, ja que o eren errors fàcils de solucionar o es va haver de reimplementar petits trossos de codi.

6.1.1 Wrapper

El wrapper és l'encarregat de la comunicació de les classes del software VBS amb les classes natives del sistema operatiu. Quan es va executar el programa per primera vegada, va sortir el següent error:

```
Startup of java application timed out. If this is due to server overload
consider increasing wrapper.startup.timeout.
```

Tal i com es pot deduir de la descripció de l'error, està relacionat amb el temps d'inicialització degut a una possible sobrecàrrega de la CPU. En el nostre cas va ser perquè en comptes d'usar un client i un servidor en màquines diferents, usàvem un sol ordinador amb dues màquines virutals. Això provocava que tot i que l'ordinador és potent ja que està executant

un Windows, la màquina virtual amb Linux i la màquina virtual amb Mac OS X a la vegada, tardés més temps en incialitzar totes les comunicacions entre client i servidor.

Per solventar aquest problema, es va modificar l'opció *"wrapper.startup.timeout"* de l'arxiu */VBSclient_64bit/conf/wrapper.conf*. El valor d'aquesta opció per defecte és de 30 segons. Això implica que passat aquest temps, si el wrapper no s'ha inicialitzat del tot es considera que l'ordinador s'ha bloquejat i per tant, s'atura l'execució. Es va descomentar la linia i se li va assignar un valor de 5000 segons, temps més que suficient perquè un ordinador lent pugui executar sense problemes el wrapper[36]:

Codi 1 Wrapper timeout

```
# Default: 30 seconds
wrapper.startup.timeout = 5000
```

6.1.2 Global IP

Abans d'inicialitzar qualsevol classe del client o servidor, el software es connecta a Internet i esbrina quina és la seva adreça IP global. Si hi ha algun problema per obtenir-la, envia una excepció i atura tot el procediment. En el nostre cas, l'error és el següent:

```
Cannot obtain global IP address of the host: connect timed out.
```

La classe es troba a */VBSeclipsePayloads/src/tools/GlobalIP.java*. La seva funcionalitat principal és connectar-se a una pàgina web i preguntar quina és l'IP del client on s'està executant. De la mateixa manera que ha passat amb l'error del wrapper a l'apartat anterior, sembla que en màquines lentes es tarda més a establir la connexió i rebre una resposta de la pàgina web. Per a solucionar-ho es va optar per incrementar el timeout i el temps de lectura de 5 a 60 segons:

Codi 2 Global IP abans de la modificació

```
connection.setConnectTimeout(5000);
connection.setReadTimeout(5000);
```

Codi 3 Global IP després de la modificació

```
connection.setConnectTimeout(60000);
connection.setReadTimeout(60000);
```

6.1.3 Instal·lació Wrapper

Abans d'executar el software a través de l'script *runConsole.sh*, s'ha d'instal·lar el daemon per assegurar-se que el servei es crea correctament al sistema operatiu. Quan es va executar l'script *installDaemon.sh* per primera vegada va sorgir el següent error [37][38][39]:

Codi 4 IOExcpetion launchctl

```
Error executing "launchctl list": java.io.IOException: Cannot run program
    "launchctl": error=2, No such file or directory
```

Investigant es va arribar a la conclusió que l'error era a la linia 110 de l'arxiu */VBSclient_64bit/src/main/java/org/rzo/yajsw/os/posix/bsd/macosx/MacOsXService.java*, on s'executa la crida de sistema *clearProperty* de Java. Això provoca que falli cada vegada el *Runtime.getRuntime().exec(...)* ja que no troba l'arxiu.

Per a solucionar el problema es substituir la comanda *clearProperty* per *getProperty* de la linia 110:

Codi 5 MacOsXService abans de la modificació

```
String java = System.clearProperty("java.home") + "/bin/java";
```

Codi 6 MacOsXService desrpés de la modificació

```
String java = System.getProperty("java.home") + "/bin/java";
```

6.1.4 Gradle

Per a poder compilar l'arxiu *MacOsXService.java* de *YAJSW* és necessari executar el *Gradle* [40] on ja hi ha configurat tots els arxius i dependències. La comanda és la següent:

Codi 7 Comanda execució script gradle.sh

```
sudo -E ./gradle.sh -s
```

A l'executar-se, compila totes les classes i les guarda als arxius *wrapper.jar* i *wrapperApp.jar* que seran executats quan el servidor i el client s'iniciin. Quan el vam executar ens va sortir el següent error [41][42][43]:

Codi 8 Error Gradle

```
import sun.security.action.GetLongAction;
VBSclient\_64bit/src/main/java/org/rzo/yajsw/
    app/WrapperManagerImpl.java:1395: error: ManagementFactory is not public
    in sun.management; cannot be accessed from outside package
    com.sun.management.HotSpotDiagnosticMXBean mb = sun.management.
    ManagementFactory.getDiagnosticMXBean();
```

El problema recau en que la classe *ManagementFactory* del paquet *sun.management* no té la funció *getDiagnosticMXBean()*. Segurament van canviar l'estructura del paquet en alguna actualització i és per això que ara no funciona. Després d'investigar les diverses classes i paquets, s'ha optat per canviar la classe: *ManagementFactory* per *ManagementFactoryHelper*, ja que aquesta última si que la té.

Codi 9 Abans de la modificació: ManagementFactory

```
com.sun.management.HotSpotDiagnosticMXBean mb =
    sun.management.ManagementFactory.getDiagnosticMXBean();
```

Codi 10 Després de la modificació: ManagementFactoryHelper

```
com.sun.management.HotSpotDiagnosticMXBean mb =
    sun.management.ManagementFactoryHelper.getDiagnosticMXBean();
```

6.1.5 Inicialització Daemon

Per iniciar el daemon manualment, s'ha d'executar l'script *startDaemon.sh* que crea una plist a la carpeta */Users/capitanclient/Library/LaunchAgents*. No obstant això, quan s'executa per primera vegada a Max OS X retorna l'error *Service VBSclient NOT started*.

Investigant el log del sistema (*var/lib/system.log*) es troba el següent:

Codi 11 Inicialització Daemon

```
BUG in libdispatch client: kevent[EVFILT\_PROC] enable: "No such file or
    directory" - 0x2
```

Això apunta a que segurament les comandes usades a Linux són lleugerament diferents a les que funcionen en Mac OS X. Es fa una mica de recerca i es troba que les que hauria d'executar el daemon són les següents [37][38][39]:

- **Preparar el servei**: launchctl load /Users/capitanclient/Library/LaunchAgents/wrapper.VBSclient.plist
- **Tancar el servei**: launchctl unload /Users/capitanclient/Library/LaunchAgents/wrapper.VBSclient.plist
- **Iniciar el servei**: launchctl start wrapper.VBSclient
- **Aturar el servei**: launchctl stop wrapper.VBSclient

A l'arxiu *MacOsXService.java* hi ha el codi que inicia el daemon. Per saber si aquest s'ha iniciat correctament, s'executa la comanda *launchctl list* abans i després de realitzar les crides i compara el resultat. No obstant això, en el sistema opeartiu Mac OS X no funciona adecuadament i retorna sempre false encara que el servei s'hagués creat correctament. Per tant, es reprogramat la funció:

Codi 12 MacOsXService.java funció Daemon

```
String list = \_utils.osCommand("launchctl list | grep
    \"wrapper."+getName()+"\" ", 5000);
String[] match = new String[] {};
match = list.split(" ");
int pid;
if (list != null && !list.isEmpty() && !match[0].equals("-"))
   pid = Integer.parseInt(match[0]);
else
   pid = 0;
return pid;
```

L'objectiu d'aquesta funció és executar la comanda *launchctl list* i usa un grep amb el nom del daemon per buscar la linia de la llista d'actius. A continuació fa un split i obté el PID.

6.1.6 Sigar

Durant l'execució del client, la classe *DatabaseHandler* mostra el següent error de linkatge:

Codi 13 UnisatisfiedLinkError Sigar

```
Exception in thread "DatabaseHandlerThread" java.lang.UnsatisfiedLinkError:
    org.hyperic.sigar.Sigar.getCpuInfoList()[Lorg/hyperic/sigar/CpuInfo;
   Trigger found: Exception
```

```
start script scripts/trayMessage.gv
error in script trayMessage.gv: no process or process not configured for
    tray icon
end script scripts/trayMessage.gv
   at org.hyperic.sigar.Sigar.getCpuInfoList(Native Method)
   at
      databaseHandler.DatabaseHandler.getSystemInfo(DatabaseHandler.java:211)
   at
      databaseHandler.DatabaseHandler.setDatabase(DatabaseHandler.java:252)
   at databaseHandler.DatabaseHandler.run(DatabaseHandler.java:144)
   at java.lang.Thread.run(Thread.java:745)
```

Tot a punta que, igual que ens ha passat nombroses vegades, no reconeix la llibreria que hi ha actualment en el projecte. A la pàgina ofcial de *hyperic* s'han trobat les següents llibreries que funcionen correctament [11]:

- libsigar-universal-macosx.dylib
- libsigar-universal64-macosx.dylib

6.1.7 SQLite

De la mateixa manera que l'error anterior, la classe *databaseHandler* envia la següent excepció :

Codi 14 NullPointerException SQLite

```
Exception in thread "DatabaseHandlerThread" java.lang.NullPointerException
Trigger found: Exception
   at databaseHandler.DatabaseHandler.setDatabase(DatabaseHandler.java:248)
   at databaseHandler.DatabaseHandler.run(DatabaseHandler.java:144)
   at java.lang.Thread.run(Thread.java:745)
[DatabaseHandlerThread] ERROR databaseHandler.DatabaseHandler - Cannot open
    the database!
```

[caption=Internal log SQLite: llibreria no carregada] Es pot veure clarament que és un problema amb la base de dades *SQLite*. Al ser un *NullPointerException* es suposa que està relacionat amb que no troba la llibreria.

Per comprovar-ho s'executa el .jar *sqlite4java.jar* amb la comanda [44]:

```
java -jar sqlite4java.jar -d
```

S'obté el següent :

Codi 15 Internal log SQLite: llibreria no carregada

```
sqlite4java 282
[sqlite] Internal: loading library
[sqlite] Internal:
   java.library.path=.../Library/Java/Extensions:/Library/Java/Extensions:/Network/Library
[sqlite] Internal: sqlite4java.library.path=null
[sqlite] Internal: cwd=.../VBSeclipsePayloads/lib/.
[sqlite] Internal: default path=.../VBSeclipsePayloads/lib
[sqlite] Internal: forced path=null
[sqlite] Internal: os.name=mac os x; os=osx
[sqlite] Internal: os.arch=x86_64
[sqlite] Internal: trying to load sqlite4java-osx-x86_64
[sqlite] Internal: cannot load sqlite4java-osx-x86_64:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-x86_64 in
   java.library.path
[sqlite] Internal: trying to load sqlite4java-osx-amd64
[sqlite] Internal: cannot load sqlite4java-osx-amd64:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-amd64 in
   java.library.path
[sqlite] Internal: trying to load sqlite4java-osx-10.4
[sqlite] Internal: cannot load sqlite4java-osx-10.4:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-10.4 in
   java.library.path
[sqlite] Internal: trying to load sqlite4java-osx
[sqlite] Internal: cannot load sqlite4java-osx:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx in java.library.path
[sqlite] Internal: trying to load sqlite4java
[sqlite] Internal: cannot load sqlite4java: java.lang.UnsatisfiedLinkError:
   no sqlite4java in java.library.path
[sqlite] Internal: trying to load sqlite4java-osx-x86_64-d
[sqlite] Internal: cannot load sqlite4java-osx-x86_64-d:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-x86_64-d in
   java.library.path
[sqlite] Internal: trying to load sqlite4java-osx-amd64-d
[sqlite] Internal: cannot load sqlite4java-osx-amd64-d:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-amd64-d in
   java.library.path
```

```
[sqlite] Internal: trying to load sqlite4java-osx-10.4-d
[sqlite] Internal: cannot load sqlite4java-osx-10.4-d:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-10.4-d in
   java.library.path
[sqlite] Internal: trying to load sqlite4java-osx-d
[sqlite] Internal: cannot load sqlite4java-osx-d:
   java.lang.UnsatisfiedLinkError: no sqlite4java-osx-d in java.library.path
[sqlite] Internal: trying to load sqlite4java-d
[sqlite] Internal: cannot load sqlite4java-d:
   java.lang.UnsatisfiedLinkError: no sqlite4java-d in java.library.path
Error: cannot load SQLite
```

Es pot comprovar com intenta obrir tota una sèrie de llibreries de les que en principi no disposem. Per ordre de prioritat es busca i es prova cada una de les llibreries que hi ha al log anterior. No obstant això, cap ha funcionat correctament excepte amb *libsqlite4java-osx-10.4.jnilib* [45].

Així que un cop afegida al projecte, es torna a executar la mateixa comanda i el sistema les carrega correctament:

Codi 16 Internal log SQLite: llibreria carregada

```
[sqlite] Internal: loading library
[sqlite] Internal:
   java.library.path=.../Library/Java/Extensions:/Library/Java/Extensions:/Network/Library/Ja
[sqlite] Internal: sqlite4java.library.path=null
[sqlite] Internal: cwd=.../VBSeclipsePayloads/lib/.
[sqlite] Internal: default path=.../VBSeclipsePayloads/lib
[sqlite] Internal: forced path=null
[sqlite] Internal: os.name=mac os x; os=osx
[sqlite] Internal: os.arch=x86_64
[sqlite] Internal: trying to load sqlite4java-osx-x86_64 from
   .../VBSeclipsePayloads/lib/libsqlite4java-osx-x86_64.dylib
[sqlite] Internal: loaded sqlite4java-osx-x86_64 from
   .../VBSeclipsePayloads/lib/libsqlite4java-osx-x86_64.dylib
[sqlite] Internal: loaded sqlite 3.8.7, wrapper 1.3
SQLite 3.8.7
```

6.2 Servidor

6.2.1 MySQL Servidor

Quan s'inicia el servidor, el primer que fa és connectar-se a la base de dades relacional MySQL per crear les taules, si no estan creades, i esperar a que el client comenci a enviar paquets. Quan es va executar l'script *runConsole.sh* per primera vegada, va sortir una excepció relacionada amb el MySQL que indicava que l'usuari i la contrassenya eren incorrectes.

El servidor té un arxiu anomenat *server.cfg* on en un principi només hi havia l'informació sobre els diferents ports que usava el client per enviar les dades. No obstant això, no hi havia per enlloc cap manera d'indicar quin és l'usuari i la contrassenya per accedir a la base de dades. Analitzant el codi del servidor, es va veure que a les linies de codi on s'hauria d'agafar l'usuari i contrassenya estava buit. Així que es van haver de realitzar diverses modificacions.

A l'arxiu server.cfg es van afegir els atributs *dbHost, dbPort, dbUsername, dbPassword*:

Codi 17 server.cfg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>ServerSettings</comment>
<entry key="serverPortRegister">7772</entry>
<entry key="serverPortFDB">7773</entry>
<entry key="serverPortUpgrade">7774</entry>
<entry key="dbHost">localhost</entry>
<entry key="dbPort">3306</entry>
<entry key="dbUsername">root</entry>
<entry key="dbPassword">vbsmysql</entry>
</properties>
```

A l'arxiu vbsServer.java s'obté l'informació del .cfg de la següent manera:

Codi 18 vbsServer.java: abans de les modificacions

```
...
fdbServerPort = Integer.parseInt(settings.getProperty("serverPortFDB",
    Integer.toString(fdbServerPort)));
```

```
registerServerPort =
    Integer.parseInt(settings.getProperty("serverPortRegister",
    Integer.toString(registerServerPort)));

upgradeServerPort =
    Integer.parseInt(settings.getProperty("serverPortUpgrade",
    Integer.toString(upgradeServerPort)));

dbHost = settings.getProperty("dbHost", "");

dbPort = settings.getProperty("dbPort", "");

dbUsername = settings.getProperty("dbUsername", "");

dbPassword = settings.getProperty("dbPassword", "");
...
```

Es pot veure clarament com els quatre últims atributs estan en blanc. És per això que es van modificar perquè s'obtinguessin correctament:

Codi 19 vbsServer.java: després de les modificacions

```
...
fdbServerPort = Integer.parseInt(settings.getProperty("serverPortFDB",
    Integer.toString(fdbServerPort)));

registerServerPort =
    Integer.parseInt(settings.getProperty("serverPortRegister",
    Integer.toString(registerServerPort)));

upgradeServerPort =
    Integer.parseInt(settings.getProperty("serverPortUpgrade",
    Integer.toString(upgradeServerPort)));

dbHost = settings.getProperty("dbHost", dbHost);

dbPort = settings.getProperty("dbPort", dbPort);

dbUsername = settings.getProperty("dbUsername", dbUsername);
```

```
dbPassword = settings.getProperty("dbPassword", dbPassword);
...
```

A més a més, després de l'obtenció dels atributs, hi ha uns if que controlen si aquests s'han llegit correctament i no són NULL. S'han hagut de modificar les condicions ja que comparaven strings amb doble igual: *dbUsername == " "*. Aquesta comparació és correcte si es fa amb enters, però si es vol comparar cadenes de text s'ha d'usar una funció de la classe String anomenada *equals*. Aquests són els canvis realitzats per arreglar totes les condicions:

Codi 20 vbsServer.java: condicions arreglades

```
...
if ((fdbServerPort == 0) || (registerServerPort == 0) || (upgradeServerPort
    == 0)) {
     throw new Exception("The config file is corrupted!");
}
if (dbHost.equals("")) {
     log.error("No dbHost");
     throw new Exception("The config file is corrupted!");
}
if (dbPort.equals("")) {
     log.error("No dbPort");
     throw new Exception("The config file is corrupted!");
}
if (dbUsername.equals("")) {
     log.error("No Username");
     throw new Exception("The config file is corrupted!");
}
...
```

6.2.2 Data Extracter Servidor

Després de solventar tots els erros anteriors, es va executar l'script per inicialitzar el servidor i va sortir aquesta excepció :

Codi 21 NoSuchMethodError DataExtracter Servidor

```
Caused by: java.lang.NoSuchMethodError:
    vbsServer.DataExtracter.<init>(Ljava/lang/String;Ljava/
    lang/String;Ljava/lang/String;Ljava/lang/String;)V
     at vbsServer.VbsServer.setupDataExtracter(VbsServer.java:314)
     at vbsServer.VbsServer.setup(VbsServer.java:236)
     at vbsServer.VbsServer.run(VbsServer.java:125)
     at vbsServer.VbsServer.main(VbsServer.java:97)
```

A primera vista tot indica que no es troba la funció inicialitzadora del Data Extracter o s'estan enviant els paràmetres malament. No obstant això, resulta que el problema és que s'han compilat algunes classes i d'altres no. Això provoca que hi hagi classes compilades per última vegada el 2014 i d'altres al 2016 i per tant hi ha incompatibilitats entre els diferents .class. Només cal recompilar totes les classes i l'error desapareix [46].

6.3 Client - jNetPcap

6.3.1 Incompatiblitat

El client és la part del VBS que més temps se li ha dedicat degut a que el principal problema de compatibilitat està en les seves classes. A continuació mostrarem com es va detectar que la llibreria jNetPcap no era compatible amb Mac OS X. A més a més, també es detallaran totes les solucions i proves que s'han aplicat per intentar fer que la classe carregués correctament llibreria.

Durant l'inicialització de les classes del client va aparèixer el següent error [47]:

Codi 22 UnsatisfiedLinkError llibreria jNetPcap (Pcap)

```
Exception in thread "PacketCapturerThread" java.lang.UnsatisfiedLinkError:
    com.slytechs.library.NativeLibrary.dlopen(Ljava/lang/String;)J
Trigger found: Exception
     at com.slytechs.library.NativeLibrary.dlopen(Native Method)
     at com.slytechs.library.NativeLibrary.<init>(NativeLibrary.java:121)
     at com.slytechs.library.JNILibrary.<init>(JNILibrary.java:433)
     at com.slytechs.library.JNILibrary.loadLibrary(JNILibrary.java:162)
     at com.slytechs.library.JNILibrary.register(JNILibrary.java:289)
     at com.slytechs.library.JNILibrary.register(JNILibrary.java:266)
     at com.slytechs.library.JNILibrary.register(JNILibrary.java:249)
     at org.jnetpcap.Pcap.<clinit>(Pcap.java:509)
     at
         packetCapturer.PacketCapturer.getValidDevices(PacketCapturer.java:216)
     at packetCapturer.PacketCapturer.run(PacketCapturer.java:151)
     at java.lang.Thread.run(Thread.java:745)
```

Si s'realitza detalladament el traceback de l'error anterior s'arribar a la següent conclusió:

A l'inicialitzar-se el client, es crida a la funció *run()* de la classe *vbsClient*. Aquesta funció és un bucle que inicialitza el sistema i totes les classes amb els seus threads cridant a la funció *startSystem()* i aquesta a través de la crida *myPacketCapturer.start()* inicia la classe *PacketCapturer*:

Codi 23 vbsClient.java: funció run()

```
...
log.info("Starting the client");
if (!startSystem()) {
```

```
  log.error("Cannot start the client");
  System.exit(1);
}
...
```

PacketCapturer inicia un thread per cada interficie. Quan s'inicia el thread de la classe *Packet Capturer*, aquesta crida a una funció interna per obtenir un llistat de totes les interfícies de xarxa de l'ordinador disponibles i vàlides per a la posterior captura de paquets:

Codi 24 PacketCapturer.java: funció getValidDevices()

```
...
while (setupInProgress) {
  supportedDevices = this.getValidDevices();
  for (PcapIf device : supportedDevices) {
    Capturer myCapturer = new Capturer(device, myPacketsQueue);
    myCapturers.add(myCapturer);
    setupInProgress = false;
  }
}
...
```

La funció *getValidDevices()* crida a la funció *findAllDevs(allDevices, errorBuffer)* de la classe *Pcap* de la llibreria *JnetPcap*.A partir d'aquí, es realitzen crides a les llibreries natives de JNI fins que envia l'excepció mostrada. Resumint, la funció del VBS que falla és *getValidDevices* de la classe *Capturer* quan aquesta fa una crida a una funció de *Pcap.java*.

En un primer moment es va creure que podia ser un problema de compatibilitat, ja que si quan es va desenvolupar aquest sofware estaven usant màquines de 32 bits, al recompil·lar-ho a la màquina virtual que és de 64 bits, podria causar algun error entre les classes recompil·lades i la llibreria *jNetPcap* o *libpcap* de 32 bits. Després de comprovar que ambdues llibreries són de 64 bits s'ha descartat aquesta possibilitat.

A continuació s'ha comprovat si el problema recau només a la funció *getValidDevices*, però no és aixi. Qualsevol funció de la llibreria jNetPcap que es crida acabava enviant una excepció.

Després d'investigar com funcionen les llibreries en Linux i Mac OS X, s'ha vist que moltes vegades és necessari declarar certes variables globals per indicar al sistema operatiu on estan les diverses llibreries. Per tant, es va buscar la manera d'indicar a la màquina virtual de Java on trobar la llibreria nativa, ja que Java no usa la variable *CLASSPATH* sinó que la

carpeta on es poden trobar s'ha de definir a *java.library.path*. Es modifica l'script *wrapper.sh*, que inicialitza el software, per afegir aquest path [48]:

Codi 25 wrapper.sh

```
-Djava.library.path=/usr/local/lib
```

Tot i així seguia apareixient el mateix error. Es va optar per modificar algunes funcions de la classe *Capturer* perquè en comptes de cridar a *Pcap.java*, crides a *PcapIf.java*. D'aquestat manera s'intenta descartar que el problema fos una classe en concret. No obstant això, després de fer un backup i realitzar els canvis corresponent, seguia sortint el mateix error però en comptes d'apuntar a la classe Pcap, apuntava a PcapIf:

Codi 26 UnsatisfiedLinkError: llibreria jNetPcap (PcapIf)

```
Exception in thread "PacketCapturerThread" java.lang.UnsatisfiedLinkError:
    com.slytechs.library.NativeLibrary.dlopen(Ljava/lang/String;)J
     Trigger found: Exception
          at com.slytechs.library.NativeLibrary.dlopen(Native Method)
          at com.slytechs.library.NativeLibrary.<init>(NativeLibrary.java:121)
          at com.slytechs.library.JNILibrary.<init>(JNILibrary.java:433)
          at com.slytechs.library.JNILibrary.loadLibrary(JNILibrary.java:162)
          at com.slytechs.library.JNILibrary.register(JNILibrary.java:289)
          at com.slytechs.library.JNILibrary.register(JNILibrary.java:266)
          at com.slytechs.library.JNILibrary.register(JNILibrary.java:249)
          at org.jnetpcap.PcapIf.<clinit>(PcapIf.java:51)
          at packetCapturer.PacketCapturer.getValidDevices(PacketCapturer
              .java:217)
          at packetCapturer.PacketCapturer.run(PacketCapturer.java:151)
```

Per tant, queda descartat que sigui només problema d'una funció o d'una classe. És problema de tota la llibreria. A partir d'aquí l'objectiu és descartar que sigui problemes de linkatge. És a dir, que les llibreries no s'estiguin carregant correctament o que estiguin linkades d'alguna manera diferent en Mac OS X.

Les llibreries natives de Mac OS X estan a la carpeta */usr/local/lib* i tenen diversos softlinks entre elles. La que ens interessa és la *libpcap*, ja que és la que s'usa per a la captura de paquets.

Es modifica el client perquè carregui les llibreries. S'afegeix una condició a l'if perquè detecti el sistema operatiu i es modifiquen les funcions de load perquè carreguin les llibreries de libpcap [49]:

Codi 27 vbsClient.java: funció loadLibraries()

```
...
else if (System.getProperty("os.name").toLowerCase().contains("mac")) {
  log.info("VBS Client is running under Mac, loadding libraries...");
  String appPath =
      URLDecoder.decode(VbsClient.class.getProtectionDomain().getCodeSource().
      getLocation().getPath(), "UTF-8").replace("client.jar", "");
  loadLibrary("/usr/local/opt/libpcap/lib/libpcap.dylib");
}
...
```

A continuació es decideix reinstal·lar la llibreria libpcap. S'ha instal·lat amb la comanda *brew* i s'han assignat aquestes FLAGS [50]:

- LDFLAGS: -L/usr/local/opt/libpcap/lib
- CPPFLAGS: -I/usr/local/opt/libpcap/include
- LD_LIBRARY_PATH: /usr/local/opt/libpcap/lib
- LYLD_LIBRARY_PATH: /usr/local/opt/libpcap/lib

Però segueix sense funcionar.

Per últim, s'intenta arribar al centre de l'error: la funció *dlopen* [51][52] de la llibreria *slytechs*. Podria ser que, en algun moment de la cadena, algun dels links estigui fallant o enviant un valor erroni. Modificant la llibreria nativa de slytechs i estudiant les seves funcions, es descobreis que el paràmetre que li arriba a la funció *dlopen* és l'string *jnetpcap*, per tant es determina que tota la cadena funciona correctament.

S'arriba a la conclusió que actualment la llibreria *jNetPcap* no està suportada per Mac OS X ja que no s'ha trobat cap manera de linkar els .so (llibreries Linux)[53][54].Així que es canvia el rumb del projecte i es decideix intentar modificar la llibreria perquè es pugui executar en aquest sistema operatiu.

6.3.2 Adaptació a Mac OS X

S'ha comprovat que no hi ha manera de fer que Mac OS X carregui i llegeixi les llibreries en el format que actualment ofereix la web oficial: llibreria en format .so (Linux) i .dll (Windows).

Totes les llibreries que s'usen en aquest projecte són llibreries dinàmiques [55], ja que d'aquesta manera es poden canviar funcions del codi o canviar llibreries sense haver de recompil·lar tot el software. Existeixen també les llibreries estàtiques (.a en Linux) [56], però ja gaire bé no s'usen.

En el cas de Mac OS X existeixen dos formats: el .dylib i el .jnilib. La primera extensió s'usava fa uns anys fins que es va decidir que en els nous Mac OS X es comencés a usar el format jnilib. A la pràctica és iniferent quin dels dos formats s'usi per a les llibreries. Tot i que Apple va decidir que la funció de càrrega de la llibreria buscaria arxius .jnilib, es tan fàcil com crear un *softlink* o canviar el nom d'extensió[57].

Crear llibreria a usant Java

La llibreria jNetPcap que usava el software VBS consta de les classes en format .java i .class. Usant JNI es creen automàticament tots els .h de cada classe on les capçaleres estan en format JNI [Codi 32]:

Codi 28 Exemple capçalera JNI

```
JNIEXPORT void JNICALL Java_ClassName_MethodName
  (JNIEnv *env, jobject obj)
 {
    //Exemple JNI
 }
```

El següent pas per a poder crear el .dylib és crear els .c o els .cpp de tal manera que les funcions d'aquests corresponguin amb les dels .h generats amb anterioritat. No obstant això. el principal problema és que això implicaria haver de reprogramar tota llibreria. Per cada arxiu Java, s'hauria de traduir funció a funció a C o C++ [58].

Es va arribar a la conclusió que era inviable el volum de feina que això comportaria pel projecte. No només pel temps, sinó perquè implicaria entendre cada una de les funcions i arriscar-se a que n'hi haguessin algunes que no fossin compatibles amb el sistema operatiu. Per tant, s'opta per una altra solució.

Crear llibreria usant C

L'única possible alternativa és trobar els arxius de les classes en format .c o .cpp. A la web oficial de *jNetPcap* [14] és possible baixar-se el codi font de la llibreria on hi ha les classes en Java i C.

Els passos a seguir per a poder crear el dylib són els següents [59]:

1. Generar els .h.
   (a) Compilar els arxius .java per obtenir els .class
   (b) Compilar els .class amb un flag de JNI per aconseguir els .h
2. Generar els .o compilant els .c.
3. Generar la llibreria dylib usant els .h i els .o.

Aquest procés ha estat verificat amb un test [60] que consisitia seguir els mateixos passos però amb un "HelloWorld" i s'obtenia una llibreria que s'executava correctament.

Usant la comanda *brew* s'ha instal·lat el programa *Ant* [61], un software que llegeix el fitxer *build.xml* i compila automàticament totes les classes executant la comanda:

Codi 29 Comanda ANT

```
ant -f build.xml
```

Retorna el següent error:

Codi 30 Error compilació jNetPcap: llibreria cpptasks

```
 Buildfile: VBSeclipsePayloads/jnetpcap-src-1.4.b0004-1/build.xml
[taskdef] Could not load definitions from resource cpptasks.tasks. It
    could not be found.
[typedef] Could not load definitions from resource cpptasks.types. It
    could not be found.

 BUILD FAILED
 VBSeclipsePayloads/jnetpcap-src-1.4.b0004-1/build.xml:116: taskdef class
     org.vafer.jdeb.ant.DebAntTask cannot be found
  using the classloader AntClassLoader[]

 Total time: 0 seconds
```

Tot indica que no troba la llibreria *cpptasks* [62]. A través de la pàgina oficial d'aquesta llibreria [63] es pot obtenir el fitxer .jar. S'ha de copiar a la carpeta on s'ha instal·lat el software Ant: */usr/local/Cellar/ant/1.9.7/libexec/lib/* [64]. A continuació surt una el següent error de compatibiltiat:

Codi 31 Error compilació jNetPcap: DebAntTask

```
Buildfile: /Users/Trelis/Dropbox/ATM/TFG/Mac/VBSeclipsePayloads/
 jnetpcap-src-1.4.b0004-1/build.xml

BUILD FAILED
/Users/Trelis/Dropbox/ATM/TFG/Mac/VBSeclipsePayloads/
 jnetpcap-src-1.4.b0004-1/build.xml:116: taskdef class
     org.vafer.jdeb.ant.DebAntTask cannot be found
 using the classloader AntClassLoader[]

Total time: 0 seconds
```

Ràpidament s'arriba a la conclusió que el problema és que aquest *build.xml* està programat perquè funcioni en un sistema Debian. Comentant les següentes linies soluciona el problema i permet la compilació de la classe:

Codi 32 build.xml

```
...
<taskdef name="deb" classpathref="ant.deb.classpath"
    classname="org.vafer.jdeb.ant.DebAntTask" />
...
```

Després de copilar la llibreria s'han obtingut els arxius .class. El següent pass és obtenir els arxius .h amb les capçaleres. Es poden aconseguir usant una comanda de Java amb un flag de JNI.

Per agilitzar la creció dels .h s'ha programat un script anomenat *createH.py* que automatitza tot el procés. Per a cada .class que hi hagi a la carpeta i subcarpetes en crea el .h i els guarda tots a una carpeta destí:

Codi 33 Script createH.py

```
from subprocess import call
import os
```

```
# traverse root directory, and list directories as dirs and files as files
for root, dirs, files in os.walk("."):
    path = root.split('/')
    for file in files:
      finalPath = ""
      boolean = 0
      for a in path:
         if boolean:
            finalPath += a + '.'
         boolean = 1
        if "class" in finalPath + file:
            file = file.split('.')[0]
            call('javah -classpath ./jnetpcap.jar:' + file + '.java -jni ' +
                finalPath + file, shell=True)

for file in os.listdir("./"):
    if ".h" in file:
        call('mv ' + file + ' includes', shell=True)
```

El següent pass és obtenir els .o. És a dir, compilar totes les classes en C. De la mateixa manera que en el pas anterior, s'ha creat un script per agilitzar el procés de compilació:

Codi 34 Script createDYLIB.py

```
from subprocess import call
import os

flags = "-I-I/System/Library/Frameworks/JavaVM.framework/Headers"
flags += " -I/System/Library/Frameworks/JavaVM.framework/Headers"
flags += " -I/System/Library/Frameworks/JavaVM.framework/Headers/darwin/"
flags += "
    -I//Users/Trelis/Dropbox/ATM/TFG/Mac/VBSeclipsePayloads/jnetpcap-src-1.4.b0004-1/build/in
flags += "
    -I//Users/Trelis/Dropbox/ATM/TFG/Mac/VBSeclipsePayloads/jnetpcap-src-1.4.b0004-1/src/c"

# traverse root directory, and list directories as dirs and files as files
for root, dirs, files in os.walk("./objectFiles/"):
    path = root.split('/')
    for file in files:
```

```
      finalPath = ""
      boolean = 0
      for a in path:
         if boolean:
            finalPath += a + '/'
         boolean = 1
        if ".o" in finalPath + file:
            file = file.split('.')[0]
            call("gcc " + flags + " -undefined dynamic_lookup -dynamiclib -o
                lib" + file + ".dylib " + finalPath + file + ".o", shell=True)
            #print 'g++ ' + flags + ' -o lib' + file + '.dylib ' + finalPath
                + file + '.cpp'

for file in os.listdir("./"):
    if ".dylib" in file:
        call('mv ' + file + ' libraries', shell=True)
```

Tot i així, s'obté el següent error:

Codi 35 Error compilació jNetPcap: packet_jscanner.h

```
src/c/packet_jscanner.h:262:11: error: field has incomplete type 'header_t
    []'
header_t pkt_headers[]; // One per header + 1 more for payload
```

L'error indica que hi ha algun problema a l'struct *packet_state_t* de la classe *packet_jscanner.h*:

Codi 36 packet_jscanner.h: struct packet_state_t

```
typedef struct packet_state_t {
   uint8_t pkt_flags;     // flags for this packet
   flow_key_t pkt_flow_key; // Flow key calculated for this packet, must be
       first
   jobject pkt_analysis;  // Java JAnalysis based object if not null
   uint64_t pkt_frame_num; // Packet's frame number assigned by scanner
   uint64_t pkt_header_map; // bit map of presence of headers

   uint32_t pkt_wirelen;  // Original packet size

   int8_t pkt_header_count; // total number of main headers found
```

```
   header_t pkt_headers; // One per header + 1 more for payload

   int8_t pkt_subheader_count; // total number of sub headers found
   header_t pkt_subheaders; // One per header + 1 more for payload
} packet_state_t;
```

El motiu de l'error és que *header_t pkt_headers[];* i *header_t pkt_subheaders[];* són arrays i per tant, han de tenir una mida definida. En C, el compilador ho permet sempre i quan només hi hagi un array d'aquest tipus i estigui al final de l'struct. D'aquesta manera, el declara com a 0 size i el va augmentant si cal. Això és possible perquè a la pila de la memòria no hi ha més codi darrere d'aquest array, així que si l'augmenta no sobrescriurà res. Ara bé, en el nostre cas tenim dos arrays sense mida. Hi ha dues possibles solucions que funcionen (s'ha testejat i ambdues funcionen correctament)[65]:

1. Eliminar les tres linies següents, ja que en cap part del codi s'utiltizen i d'aquesta manera s'aconsegueix que *header_t pkt_headers;* sigui l'últim element de l'struct:

   Codi 37 packet_jscanner.h: struct packet_state_t solució 1

   ```
   int8_t pkt_header_count; // total number of main headers found
   header_t pkt_headers[]; // One per header + 1 more for payload
   int8_t pkt_subheader_count; // total number of sub headers found
   ```

2. L'altre solució i la que s'ha acabat escollint, és que en comptes de declarar els arrays amb una mida determinada, a l'struct es declaren els punters. D'aquesta manera no cal indicar a l'inicialització quina és la mida de l'array:

   Codi 38 packet_jscanner.h: struct packet_state_t solució 2

   ```
   header_t *pkt_headers; // One per header + 1 more for payload
   header_t *pkt_subheaders; // One per header + 1 more for payload
   ```

Cal esmentar que aquests canvins només s'han hagut de fer perquè el compilador es queixa si s'executa en sistemes Mac OS X. Si es compila en Linux no s'ha de fer cap modificació de la llibreria.

El problema amb el que ens hem topat en aquest punt és que tot i tenir tots els .dylib, un per classe, no hi ha manera d'ajuntar-los tots per fer-ne un de sol [66]. Tot i carregar totes les llibreries correctament [49], a l'hora de compilar el client seguien sortint errors de

linkatge. Per això, s'ha arribat a la conclusió que per a poder executar la llibreria jNetPcap en un sistema operatiu Mac OS X no serveix usar JNI per crear el dylib, sinó que s'hauria de programar des de 0 o detectar quines són les classe que no són compatibles.

Per aquestes raons, s'ha cregut convenient buscar una alternativa, ja que voler reprogramar la llibreria jNetPcap s'allunya molt a l'objectiu principal del projcte.

6.4 Client - Pcap4j

L'alternativa a que s'ha trobat és la llibreria *Pcap4j* [26], creada justament perquè llibreries com *jNetPcap* no oferien totes les funcionalitats i no podien ser executades en Mac OS X.

En aquest apartat s'explica com es configura l'entorn per a poder treballar amb la llibreria *Pcap4j*, es mostren tots els canvis duts a terme per adaptar el software VBS i es detallen tots els canvis realitzats a les classes del client.

6.4.1 Instal·lació

Tota l'informació respecte la llibreria es pot trobar a la pàgina de Github [26]. Abans de res, s'han d'instal·lar totes les llibreries que Pcap4j necessita. Per aquest projecte s'han usat les següents versions:

- **jna-4.4.2 [67]:** Primer de tot s'ha d'obtenir el fitxer .jar amb la llibreria i guardar-lo a la carpeta del client. A continuació s'ha de verificar que la versió de jna que hi ha al fitxer *pom.xml*, de la carpeta de la llibreria Pcap4j, sigui la mateixa versió:

  Codi 39 pom.xml

```
...
<dependency>
  <groupId>net.java.dev.jna</groupId>
  <artifactId>jna</artifactId>
  <version>4.2.2</version>
</dependency>
...
```

  A més a més, ens hem s'assegurar que la llibreria *libjnidispatch.jnilib* estigui a la carpeta *App* del client i a */usr/local/lib*.

  Per últim, s'ha de modificar el wrapper, si s'escau, per afegir els següents flgs:

  Codi 40 wrapper.sh: flags

```
-Djna.library.path=/usr/local/lib
-Djna.nosys=true
-Djna.nounpack=true
```

Si alguna màquina té alguna altra versió de JNA, podria ser que no funcionés correctament amb la llibreria Pcap4j. Si fos així, segurament s'obtendria un error similar al següent:

Codi 41 Excepció JNA

```
Exception in thread "PacketCapturerThread"
    java.lang.UnsatisfiedLinkError: Can't obtain updateLastError method
    for class com.sun.jna.Native
Trigger found: Exception
start script scripts/trayMessage.gv
end script scripts/trayMessage.gv
   at com.sun.jna.Native.initIDs(Native Method)
   at com.sun.jna.Native.<clinit>(Native.java:148)
   at com.sun.jna.Pointer.<clinit>(Pointer.java:41)
   at
       com.sun.jna.ptr.PointerByReference.<init>(PointerByReference.java:28)
   at
       com.sun.jna.ptr.PointerByReference.<init>(PointerByReference.java:24)
   at org.pcap4j.core.Pcaps.findAllDevs(Pcaps.java:48)
   at
       packetCapturer.PacketCapturer.getValidDevices(PacketCapturer.java:225)
   at packetCapturer.PacketCapturer.run(PacketCapturer.java:160)
   at java.lang.Thread.run(Thread.java:745)
```

- **slf4j-api-1.7.21 i slf4j-nop-1.7.21:** Els jar es poden obtenir de la web oficial [68] i de la carpeta del projecte.
- **libpcap** La llibreria *libpcap* es pot instal·lar fàcilment a través de la comanda *brew* [69].

6.4.2 Classe Packet Capturer

Aquesta classe s'encarrega de gestionar la classe *Capturer*. Detecta les interficies obertes i disponibles per capturar paquets del client i, per cada una, crea un thread de la classe *Capturer*. A través d'un bucle, monitoritza cada un dels threads fins que aquests acaben. Llavors s'assegura que la classe *Flow Generator* agafa els packets que s'han capturat.

Funció run

Primer de tot detecta quins són les interfícies que estan obertes i disponibles del client a través de la funció *getValidDevices*. Un cop obté l'array de interficies vàlides, crea un thread per cada una d'elles, els inicialitza i s'assegura que s'hagin inicialitzat correctament.

Codi 42 PacketCapturer.java: funció run()

```
public void run() {
    log.info("Starting thread");
    boolean setupInProgress = true;
    while (setupInProgress) {
        supportedDevices = this.getValidDevices();

        for (PcapNetworkInterface device : supportedDevices) {
            Capturer myCapturer = new Capturer(device, myPacketsQueue);
            myCapturers.add(myCapturer);
            setupInProgress = false;
        }
    }
    for (Capturer currentCapturer : myCapturers) {
        currentCapturer.start();
        while (true) {
            if (currentCapturer.isAlive()) {
                break;
            } else {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                }
            }
        }
    }
    this.isStarted = true;
...
```

A continuació es queda en un bucle infinit monitoritzant tots els threads. Si algun d'ells s'atura a causa d'algun error, els para tots i envia una excpeció a la classe *VbsClient* perquè es reinici.

Codi 43 PacketCapturer.java: continuació funció run()

```
...
    runFlag = true;
    boolean stopping = false;
    while (runFlag) {
        try {
            if (this.addedDevices() && !stopping) {
                log.info("New network devices were added. Shutting down the
                    PacketCapturer to enforce client restart");
                stopping = true;
                this.stop();
            }
        if (!this.isOK() && !stopping) {
            log.info("One of the capturers crashed. Shutting down the
                PacketCapturer to enforce client restart");
            stopping = true;
            this.stop();
        }
        Thread.sleep(10000);
        } catch (InterruptedException e) {
        }
    }
    log.info("Exiting thread PacketCapturer");
}
```

Funció getValidDevices

Aquesta és la funció principal de la classe *Packet Capturer*. El tipus de la classe dels dispositius que es capturen són *PcapNetworkInterface*, de la llibreria Pcap4j. Aquesta classe conté el següents atributs i funcions [70]:

- **boolean equals(Object obj)**: comparació de dispositius.
- **List<PcapAddress> getAddresses()**: obté totes les adeces que té un dispositiu.
- **String getDescription()** : descripció del dispositiu.
- **ArrayList<LinkLayerAddress> getLinkLayerAddresses()** : obté totes les adreces MAC del dispostiu.

- **String getName()** : obté el nom del dispositiu.
- **int hashCode()** : indica si té codi hash.
- **boolean isLocal()** : indica si és una interficie local.
- boolean **isLoopBack()** : indica si és una interficie de loopback.
- **PcapHandle openLive(int snaplen, PcapNetworkInterface.PromiscuousMode mode, int timeoutMillis)** : aquesta funció permet l'inicialització del handler per capturar paquets. S'explica més detalladament a l'apartat on s'explica la classe *Capturer*.
- String **toString()**: converteix qualsevol dels atributs a string.

Es crea un array anomenat *validDevices* on s'emmagatzaran els dispositius que són considerats vàlids per a la captura de paquets. Per altra banda, a l'array *allDevices* es guardaran tots els dispositius que es capturen abans d'aplicar qualsevol filtre.

A través de la classe *Pcaps* s'obtenen tots els dispositius i es guarden a l'array *allDevices*. A continuació s'aplica el filtre. Per a cada dispositiu s'exclouen tots els que tenen adreces de loopback i tots els que no tenen una adreça IP que no sigui del tipus IPv4:

Codi 44 PacketCapturer.java: funció getValidDevices()

```
/**
* Gets list of supported network devices.
* @return list of supported network devices
*/
private ArrayList<PcapNetworkInterface> getValidDevices() {
   ArrayList<PcapNetworkInterface> validDevices = new
       ArrayList<PcapNetworkInterface>();
   List<PcapNetworkInterface> allDevices = new
       ArrayList<PcapNetworkInterface>();
   try {
      allDevices = Pcaps.findAllDevs();
   } catch (PcapNativeException e) {
      log.info(e.getMessage());
   }
   for (PcapNetworkInterface device : allDevices) {
      /* This ifs allows to exclude loopback addresses */
      if (device.getName() != null) {
         if (device.getName().toLowerCase().contains("lo")) {
```

```
            continue;
         }
         if (device.getName().toLowerCase().contains("any")) {
            continue;
         }
         if (device.getName().toLowerCase().contains("usbmon")) {
            continue;
         }
      }
      if (device.getDescription() != null) {
         if (device.isLoopBack()) {
            continue;
         }
      }
      List<PcapAddress> addresses = device.getAddresses();
      boolean validAddress = false;
      for (PcapAddress address : addresses) {
         String a = address.getAddress().toString();
         if (address.getAddress() instanceof Inet4Address)
         validAddress = true;
      }
      if (!validAddress)
         continue;
      validDevices.add(device);
   }
   return validDevices;
}
```

Si un dispositiu passa tots els filtres, s'afageix a l'array *validDevices* i es retorna perquè es puguin crear els threads.

A continuació es mostren uns exemples de dues interficies del client. La primera és un dispositiu vàlid per a la captura de paquets i la segona és un dispositiu loopback:

Codi 45 Exemple interficie vàlida

```
name: [en0]
description: [null]
address: [/fe80:0:0:0:20c:29ff:fe5c:cc87]
address: [/192.168.59.129]
```

```
loopBack: [false]]
local: [true]
```

Com es pot apreciar, el dispositiu *en01* és una interficie amb una adreça MAC però també una adreça IPv4. A més a més, s'indica clarament que no és loopback.

En canvi, el següent dispositiu es pot comprovar que té l'atribut loopBack com a true i que l'adreça IP és la del propi ordinador. Per tant, aquesta interficie està considerada com a no vàlida.

Codi 46 Exemple interficie no vàlida

```
name: [lo0]
description: [null]
address: [/0:0:0:0:0:0:0:1]
address: [/127.0.0.1]
address: [/fe80:0:0:0:0:0:0:1]
loopBack: [true]]
local: [true]
```

6.4.3 Classe Capturer

L'objectiu d'aquesta classe consisteix en capturar tots els paquets que passen per una interficie. En aquest projecte s'ha hagut de modificar tota ja que depén completament de la llibreria *Pcap4j*. A continuació s'explica el funcionament de cada una de les funcions.

Packet Capturer és l'encarregat de crear un thread d'aquesta classe per interficie. A la constructora se li ha d'enviar l'interficia des d'on es vol que es capturin paquets i un array on s'emmagatzeran tots els paquets considerats vàlids que serà usat més endavant per la classe *Flow Generator* per crear les traces.

Constructora

Codi 47 Capturer.java: constructora

```
public Capturer(PcapNetworkInterface device, CapturedPacketsQueue
    myPacketQueue){
   log.info("Creating thread Capturer");
   this.device = device;
   this.myPacketQueue = myPacketQueue;
```

```
   String interfaceName;
   if (device.getDescription() == null) {
      interfaceName = device.getName();
   } else {
      interfaceName = device.getDescription();
   }
   capturerThread = new Thread(this);
   capturerThread.setName("CapturerThread [" + interfaceName + "]");
   capturerThread.setPriority(Thread.MIN_PRIORITY);
   log.info("Capturer setup is finished for INTERFACE: [" + device.getName()
       + " (" + device.getDescription() + ")]");
}
```

Funció Run

Aquesta és la funció principal de la classe. Consta de diverses parts i conté atributs de la llibreria *Pcap4j*, per això s'explicarà per seccions i es detallaran tots els tipus de la llibreria. S'explicaran les funcions de la llibreria més importants i s'obviaran les bàsiques.

L'idea principal és crear un *PcapHandle* [71] amb uns certs atributs i crear un bucle amb un *listener* que es quedi escoltant la interficie. Si un paquet és capturat, la funció *gotPacket* el tractarà.

Primer de tot, es comprova quina de les IPs que té l'interficie és vàlida. S'extreu l'adreça IP, la màscara i es guarda a la variable *filterSubnet* part del filtre que s'usarà més endavant per acotar els paquets que es capturen per evitar soroll.

Codi 48 Capturer.java: funció run() - Filtre

```
List<PcapAddress> addresses = device.getAddresses();
boolean validAddress = false;
String filterSubnet = "";
InetAddress ip = null;
for (PcapAddress address : addresses) {
   if (address.getAddress() instanceof Inet4Address) {
      IPAddress = address.getAddress().getAddress();
      byte[] subnetIP = IPAddress.clone();
      byte[] subnetMask = address.getNetmask().getAddress();
      mask = (Inet4Address)address.getAddress();
      int counter = 0;
```

```
      for (byte bt : subnetMask) {
         subnetIP[counter] = (byte) (subnetIP[counter] & bt);
         counter++;
      }
      filterSubnet += " or " +
          FormatConverter.decimalFormat(subnetIP) + " mask " +
          FormatConverter.decimalFormat(subnetMask);
      IPAddresses.add(IPAddress.clone());
      validAddress = true;
      ip =
          InetAddress.getByName(FormatConverter.decimalFormat(IPAddress).toString());
      log.info("Trying to start Capturer on IP address [" +
          FormatConverter.decimalFormat(IPAddress) + "], subnet address [" +
          FormatConverter.decimalFormat(subnetIP) + "], subnet mask [" +
          FormatConverter.decimalFormat(subnetMask) + "]");
   }
}
```

A continuació és crea el *PcapHandle*. Aquesta classe de la llibreria *Pcap4j* és l'encarregada de crear el canal amb l'interficie per a la captura de paquets.

Primer de tot, amb l'IP que s'ha validat a la constructora es crea una variable anomenada *nif* del tipus *PcapNetworkInterface* [70]:

Codi 49 Capturer.java: funció run() - Obtenció IP interficie

```
PcapNetworkInterface nif = Pcaps.getDevByAddress(ip);
if (nif == null)
   return;
```

La classe *PcapHandle* té una funció anomenada *openLive* que donats uns atributs, crea el handle. Està definida com:

Codi 50 Definició PcapHandle: funció openLive

```
public PcapHandle openLive(int snaplen,
                PcapNetworkInterface.PromiscuousMode mode,
                int timeoutMillis)
                  throws PcapNativeException
```

Els paràmetres que necessita són:

- **snaplen:** nombre de bytes capturats per cada paquet.
- **mode:** indica si el mode es *promiscuous* o no. És a dir, serveix per definir si es volen capturar paquets que passen per l'interficie però no van dirigits al client, sinó que van de pas.
- **timeoutMillis:** quan es capturen els paquets es guarden a un buffer fins que són tractas. Quan el temps s'esgota o el buffer s'omple, els paquets es descarten.

En el projecte s'han definit els atributs de la següent manera:

Codi 51 Capturer.java: funció run() - Configuració Handle

```
private static final String READ_TIMEOUT_KEY = Capturer.class.getName() +
    ".readTimeout";
 private static final int READ_TIMEOUT =
     Integer.getInteger(READ_TIMEOUT_KEY, 10000); // [ms]

private static final String SNAPLEN_KEY = Capturer.class.getName() +
    ".snaplen";
private static final int SNAPLEN = Integer.getInteger(SNAPLEN_KEY, 65536);
    // [bytes]

PromiscuousMode mode = PromiscuousMode.NONPROMISCUOUS;
```

La crida de la funció *openLive* per a crear el handler és la següent:

Codi 52 Capturer.java: funció run() - Creació Handle

```
PcapHandle handle;
handle = nif.openLive(SNAPLEN, PromiscuousMode.NONPROMISCUOUS, READ_TIMEOUT);
if (handle == null || !handle.isOpen()) {
   log.error("Error while opening device for capture: " +
       errorBuffer.toString());
   log.info("Exiting thread");
   return;
}
```

L'últim que s'ha de fer abans de capturar els paquets és la configuració del filtre del handler. Tot i ser opcional, va molt bé per evitar capturar paquets que no interessen. Es fa a través de la funció *setFilter*:

Codi 53 Definició PcapHandle: funció setFilter

```
public void setFilter(String bpfExpression,
            BpfProgram.BpfCompileMode mode)
              throws PcapNativeException,
                    NotOpenException
```

Els paràmetres són els següents:

- **bpfExpression:** és un string amb totes les condicions que té el filtre.
- **mode:** Permet indicar si es vol una optimització del filtre o no.

En el nostre cas aquests són els valors dels paràmetres:

Codi 54 Capturer.java: funció run() - Atributs filtre

```
String expression = "ip and not broadcast and not multicast and not tcp port
    7773 and not (src net (10 or 172.16/12 or 192.168/16" + filterSubnet +
    ") and dst net (10 or 172.16/12 or 192.168/16" + filterSubnet + "))";

BpfCompileMode compileMode = BpfCompileMode.NONOPTIMIZE;
```

I l'expressió del filtre transformada a string és la següent:

Codi 55 Expressió filtre

```
Filtering Expression: [ip and not broadcast and not multicast and not tcp
    port 7773 and not (src net (10 or 172.16/12 or 192.168/16 or 192.168.1.0
    mask 255.255.255.0) and dst net (10 or 172.16/12 or 192.168/16 or
    192.168.1.0 mask 255.255.255.0))]
```

Com es pot comprovar en el codi anterior, l'expressió filtre tots els paquets que tinguin com adreça origen broadcast, multicast, d'IPs locals i provinent del port 7773 que és el que s'usa per comunicar-se amb el servidor.

Aquesta és la funció per configurar el filtre:

Codi 56 Capturer.java: funció run() - Creació filtre

```
handle.setFilter(expression, compileMode);
```

Un cop s'ha configurat i obert el handler, el programa ha creat la connexió amb l'interficie i està preparat per a rebre paquets. Per a poder-ho fer s'ha de definir un *Listener* [72] que es quedarà escoltant i una funció dins d'aquest per a tractar els paquets capturats:

Codi 57 Capturer.java: funció run() - Listener i funció gotPacket

```
PacketListener listener = new PacketListener() {
   public void gotPacket(Packet packet) {
      Timestamp ts = handle.getTimestamp();
      IpV4Packet ipv4p = null;
      if (packet.contains(IpV4Packet.class)) {
         ipv4p = packet.get(IpV4Packet.class);
      }
      else {
         log.debug("No IPv4 Packet");
      }
      boolean worked = false;
      worked = vbsPacket(packet, ts.getTime());
      if (!worked) {
         log.debug("Not a valid packet");
         return;
      }
      else
         log.debug("Packed Processed");
   }
};
```

En aquest punt, el *listener* només s'ha definit. Més endavant s'ha d'enviar al handler i crear el bucle.

La funció *gotPacket* es crida cada vegada que un paquet és capturat. Rep el paquet, comprova que és IPv4 i crida a la funció *vbsPacket*, que es detalla en el següent apartat, que s'encarrega de processar el paquet i guardar-lo a l'array de vàlids.

Per últim, només falta definir el bucle i configurar el handle perquè usi el Listener per a escoltar els paquets. Per a dur-ho a terme s'ha escollit usar la llibreria de Java *java.util.concurrent.Executors* [73]. La seva funcionalitat és crear una *Task* [74] a la que se li assigna el *ExecutorService*. Per això, a la tasca se li passa el handle, el listener i el thread anomenat pool:

Codi 58 Capturer.java: funció run() - Tasca

```
ExecutorService pool = Executors.newSingleThreadExecutor();
Task t = new Task(handle, listener, pool);
pool.execute(t);
```

És aquesta crida l'encarregada de crear el bucle infinit tal i com es detalla al següent aparta. Per acabar només cal deixar la funció *run* en un bucle fins que el handle es tanqui:

Codi 59 Capturer.java: funció run() - Bucle de control

```
while(handle.isOpen());
log.info("Exitting Capturer Thread");
```

Aquest bucle no seria necessari si s'hagués optat per un while dins de la funció *run* per a realitzar la captura de paquets, però com que s'ha creat un thread a part s'ha d'obligar a la funció a no acabar, ja que sinó *Packet Capturer* ho detectarà i enviarà una excepció a *VbsClient*.

Thread Captura Paquets

Per a realitzar el bucle per a escoltar i capturar paquets, s'ha creat una classe anomenada *Task*:

Codi 60 Capturer.java: classe Task

```
private class Task implements Runnable {
    private PcapHandle handle;
    private PacketListener listener;
    private ExecutorService pool;
    public Task(PcapHandle handle, PacketListener listener, ExecutorService
        pool) {
      this.handle = handle;
      this.listener = listener;
      this.pool = pool;
    }
    public void run() {
      try {
        handle.loop(COUNT, listener);
      } catch (PcapNativeException e) {
        e.printStackTrace();
      } catch (InterruptedException e) {
```

```
        e.printStackTrace();
      } catch (NotOpenException e) {
        e.printStackTrace();
      }
      finally {
         if (handle != null && handle.isOpen()) {
            log.info("Closing hanle: " + capturerThread.isAlive());
            try {
               handle.breakLoop();
            } catch (Exception e) {
               log.error("Couldn't stop the Loop");
            }
           handle.close();
         }
         if (pool != null && !pool.isShutdown()) {
            log.info("Closing Pool");
           pool.shutdown();
         }
      }
    }
  }
```

Com es pot observar en el codi anterior, primer de tot es reben els paràmetres enviats per la funció *run* de la classe *Capturer*: el handle, el listener i el thread pool.

Per a crear el bucle s'usa la funció *loop* de la classe *PcapHandle*:

Codi 61 Defnició PcapHandle: funció loop

```
public void loop(int packetCount,
        PacketListener listener)
          throws PcapNativeException,
                 InterruptedException,
                 NotOpenException
```

Els paràmetres són:

- **packetCount:** Quantitat de paquets que es volen capturar. Si aquest paràmetre és -1 el bucle serà infinit fins que l'usuari l'aturi amb una *InterruptedException*:

Codi 62 Definició PcapHandle: funció loop - Atributs

```
private static final String COUNT_KEY = Capturer.class.getName() +
    ".count";
private static final int COUNT = Integer.getInteger(COUNT_KEY, -1);
```

- **listener:** Aquest paràmetre indica quina és la funció que s'ha de cridar constantment durant el bucle. En el nostre cas és la funció *Listener* que s'ha declarat dins del *run*.

Quan s'acaba el bucle es tanca el handle i la pool.

Funció VbsPacket

Aqueta funció s'encarrega de convertir el paquet rebut en un paquet VBS de la classe *CapturedPacked*. Agafa cada atribut del paquet capturar i els assigna als atributs del paquet VBS.

Abans d'entrar en detall amb el codi, cal recordar quin format tenen els paquets que es reben. En el nostre cas, són paquets IPv4 [74] que estan dividits en *Header* i *Payload*. Al *Payload* hi ha tot el contingut del paquet. En canvi, en el *Header* [75] hi ha informació sobre les diferents capçaleres.

Per exemple, un paquet TCP està encapsulat de la següent manera:

- **Ethernet Header:** és la capa més baixa del model *OSI*. Conté informació sobre les adreces físiques MAC de l'origen i el destinatari.
- **IPv4 Header:** aquest encapçulament conté informació sobre les adreces IP de l'origen i el destinatari. A més a més, inclou més informació com el TTL, Protocol, els Flags...
- **TCP Header:** per últim hi ha l'encapsulament TCP on obtenim informació del port d'origen i de destí. També hi ha informació sobre diversos flags i el nombre de seqüència (els paquets TCP es poden partir) d'entre d'altres.

Aquest és un paquet capturat mentre es realitzaven proves de funcionament on es veu clarament els tres encapsulaments:

Codi 63 Exemple paquet TCP

```
packetlength: 66
[Ethernet Header (14 bytes)]
  Destination address: 68:a8:6d:0c:30:54
  Source address: 64:66:b3:0b:94:f2
```

```
  Type: 0x0800 (IPv4)
[IPv4 Header (20 bytes)]
  Version: 4 (IPv4)
  IHL: 5 (20 [bytes])
  TOS: [precedence: 0 (Routine)] [tos: 0 (Default)] [mbz: 0]
  Total length: 52 [bytes]
  Identification: 54397
  Flags: (Reserved, Don't Fragment, More Fragment) = (false, true, false)
  Fragment offset: 0 (0 [bytes])
  TTL: 64
  Protocol: 6 (TCP)
  Header checksum: 0xe2ab
  Source address: /192.168.1.39
  Destination address: /192.168.1.35
[TCP Header (32 bytes)]
  Source port: 39090 (unknown)
  Destination port: 17500 (unknown)
  Sequence Number: 2775134970
  Acknowledgment Number: 747291656
  Data Offset: 8 (32 [bytes])
  Reserved: 0
  URG: false
  ACK: true
  PSH: false
  RST: false
  SYN: false
  FIN: false
  Window: 4623
  Checksum: 0xccea
  Urgent Pointer: 0
  Option: [Kind: 1 (No Operation)]
  Option: [Kind: 1 (No Operation)]
  Option: [Kind: 8 (Timestamps)] [Length: 10 bytes] [TS Value: 7126785] [TS
     Echo Reply: 647202850]
```

Un exemple del payload d'un paquet capturat mentre el client es connectava a la pàgina web www.upc.edu:

Codi 64 Exemple payload paquet TCP - Hexadecimal

```
47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77
    2e 75 70 63 2e 65 64 75 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65
    65 70 2d 61 6c 69 76 65 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f 6c 3a
    20 6d 61 78 2d 61 67 65 3d 30 0d 0a 55 70 67 72 61 64 65 2d 49 6e 73 65
    63 75 72 65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a 55 73 65 72 2d 41
    67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 4d 61 63 69 6e
    74 6f 73 68 3b 20 49 6e 74 65 6c 20 4d 61 63 20 4f 53 20 58 20 31 30 5f
    31 31 5f 35 29 20 41 70 70 6c 65 57 65 62 4b 69 74 2f 35 33 37 2e 33 36
    20 28 4b 48 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b 6f 29 20 43 68 72
    6f 6d 65 2f 35 33 2e 30 2e 32 37 38 35 2e 38 39 20 53 61 66 61 72 69 2f
    35 33 37 2e 33 36 0d 0a 41 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d
    6c 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 74 6d 6c 2b 78 6d 6c 2c
    61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 3b 71 3d 30 2e 39 2c 69 6d
    61 67 65 2f 77 65 62 70 2c 2a 2f 2a 3b 71 3d 30 2e 38 0d 0a 41 63 63 65
    70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 2c 20 64 65 66 6c 61
    74 65 2c 20 73 64 63 68 0d 0a 41 63 63 65 70 74 2d 4c 61 6e 67 75 61 67
    65 3a 20 65 73 2d 45 53 2c 65 73 3b 71 3d 30 2e 38 2c 63 61 3b 71 3d 30
    2e 36 2c 65 6e 3b 71 3d 30 2e 34 0d 0a 43 6f 6f 6b 69 65 3a 20 49 31 38
    4e 5f 4c 41 4e 47 55 41 47 45 3d 22 63 61 22 3b 20 73 65 72 76 65 72 69
    64 3d 7a 38 33 30 31 3b 20 5f 67 61 74 5f 55 41 2d 36 39 33 33 35 33 36
    30 2d 31 3d 31 3b 20 5f 67 61 74 5f 55 41 2d 31 31 36 36 32 32 33 38 2d
    31 3d 31 3b 20 5f 67 61 3d 47 41 31 2e 32 2e 31 36 39 34 34 38 39 39 33
    38 2e 31 34 37 31 38 38 39 37 35 31 0d 0a 0d 0a
```

Tot i que està en hexadecimal, usant usant un convertidor es pot veure que aquest paquet és l'inici de la comunicació [76]:

Codi 65 Exemple payload paquet TCP - Traduït

```
GET / HTTP/1.1
Host: www.upc.edu
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.89 Safari/537.36
Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
```

```
Accept-Language: es-ES,es;q=0.8,ca;q=0.6,en;q=0.4
Cookie: I18N_LANGUAGE="ca"; serverid=z8301; _gat_UA-69335360-1=1;
    _gat_UA-11662238-1=1; _ga=GA1.2.1694489938.1471889751
```

La classe *CapturedPacket* té els següents atributs (s'han obviat les funcions ja que totes són de "set" i "get":

Codi 66 CapturedPacked.java: Atributs

```
/** Local IP address of the packet. */
private String localIP = "";
/** Remote IP address of the packet. */
private String remoteIP = "";
/** Local transport layer port of the packet. */
private int localPort = 0;
/** Remote transport layer port of the packet. */
private int remotePort = 0;
/** Transport layer protocol name. */
private String protocolName = "";
/** HTTP content-type (if this is an HTTP packet). */
private String contentType = "";
/** HTTP URL (if this is an HTTP packet). */
private String url = "";
/** HTTP referer (if this is an HTTP packet). */
private String referer = "";
/** Direction of the packet. I = inbound, O = outbound. */
private char packetDirection;
/** Size of the packet in Bytes. */
private int packetSize = 0;
/** Timestamp of the packet. Counted in microseconds. */
private long timestamp;
/** TCP flag CWR (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagCWR = 0;
/** TCP flag ECN (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagECN = 0;
/** TCP flag URG (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagURG = 0;
/** TCP flag ACK (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagACK = 0;
```

```
/** TCP flag PSH (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagPSH = 0;
/** TCP flag RST (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagRST = 0;
/** TCP flag SYN (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagSYN = 0;
/** TCP flag FIN (if this is a TCP packet). 0 = not exists, 1 = exists. */
private int flagFIN = 0;
/** The IP payload of the packet. */
byte[] payload = new byte[]{0};
```

Per a poder obtenir els diferents camps del paquet, aquest s'ha d'anar desclossant. És a dir, per obtenir les adreces IP primer s'ha d'obtenir el header IPv4. I si es volen obtenir els ports, primer s'ha d'obtenir el paquet IPv4 i sobre aquest obtenir el header TCP.

Es comença obtenint el payload, el paquet IPv4 i el header IPv4. S'ha obviat l'encapsulament Ethernet ja que per aquest projecte no ens interessa:

Codi 67 Capturer.java: funció vbsPacket() - packet IPv4

```
...
final IpV4Packet packetIPv4 = packet.get(IpV4Packet.class);
final IpV4Header headerIPv4 = packetIPv4.get(IpV4Packet.class).getHeader();
final Packet payload = packetIPv4.get(IpV4Packet.class).getPayload();
log.debug("Creating VBS Packet");
   if (headerIPv4 == null)
      return false;
...
```

A continuació és crea el paquet VBS i es comencen a assignar els tributs a mesura que es van obtenint. Els primers en assignar-se són els atributs obtinguts per la capçalera i els de l'encapsulament més elevat. En aquest cas l'IP.

Codi 68 Capturer.java: funció vbsPacket() - Atributs packet VBS

```
...
CapturedPacket newPacket = new CapturedPacket();
newPacket.setTimestamp(timestamp);
newPacket.setPacketSize(packet.length()); //bytes
newPacket.setProtocolName("IP");
newPacket.setPayload(payload.getRawData()); //byte[]
```

```
...
```

Les adreces IP de l'origen i el destinatari s'obtenen del header IPv4. Usant les funcions *getSrcAddr()* i *getDstAddr()* s'obtenen en el format de Java *Inet4Address* [77] i s'usa la funció *getAddress()* per a convertir-les en un array de bytes:

Codi 69 Capturer.java: funció vbsPacket() - IP

```
...
byte[] sourceIP = headerIPv4.getSrcAddr().getAddress();
byte[] destinationIP = headerIPv4.getDstAddr().getAddress();
...
```

Ens les paquets VBS també s'ha de definir la direcció per a facilitar la creació de les traces. És a dir, si el paquet s'envia des del client, aquest atribut se l'hi assignarà el caràcter O, però si és el client qui rep el paquet, el caràcter serà I. En canvi, si cap de les adreces fos local el paquet s'exclouria, ja que es considera soroll:

Codi 70 Capturer.java: funció vbsPacket() - Direcció del paquet

```
...
if (isLocalIP(sourceIP)) {
   newPacket.setPacketDirection('O');
   newPacket.setLocalIP(FormatConverter.decimalFormat(sourceIP));
   newPacket.setRemoteIP(FormatConverter.decimalFormat(destinationIP));
}
else if (isLocalIP(destinationIP)) {
   newPacket.setPacketDirection('I');
   newPacket.setLocalIP(FormatConverter.decimalFormat(destinationIP));
   newPacket.setRemoteIP(FormatConverter.decimalFormat(sourceIP));
}
else {
   if (stateOK) {
      String interfaceIPAddresses = "";
      for (byte[] bar : IPAddresses) {
         if (interfaceIPAddresses != "") {
            interfaceIPAddresses += ", ";
         }
         interfaceIPAddresses += FormatConverter.decimalFormat(bar);
      }
```

```
      log.debug("Received packet has no local IP address. IP Addresses
          contained by the packet: source = " +
FormatConverter.decimalFormat(sourceIP) +
", destination = " +
FormatConverter.decimalFormat(destinationIP) +
". Probably IP address changed on the interface, because the already
    registered IP addresses on the interface are: " +
interfaceIPAddresses);
   }
return false;
}
...
```

Ja per acabar, només queden per assignar els atributs relacionats amb el següent nivell d'encapsulament: els ports. A l'exemple anterior hem vist un paquet *TCP*, però no és l'únic tipus que es pot capturar. En aquest projecte se'n tenen en compte quatre: *TCP*, *UDP*, *ICMP* i *HTTP*. Degut a que la classe *Pcap4j* no el soporta, s'ha hagut d'excloure el protocol *HTTP*. S'hauria de crear el protocol des de 0 i incloure'l a la llibreria, però per falta de temps, no s'ha pogut realtizar aquesta contribució.

Per a determinar quin tipus d'encapsulament té el paquet, s'usa el camp *protocol* del *Header IPv4*:

```
...
IpNumber protocol = headerIPv4.getProtocol();
String p = protocol.valueAsString();
...
```

La funció retorna un valor numèric que determina el protocol. En el nostre cas només tenim en compte els següents [78]:

- **Transmission Control (TCP)**: 6
- **User Datagram (UDP)**: 17
- **Internet Control Message (ICMPv4)**: 1

Si és *TCP*, s'obtenen els ports, es determina si són locals o remots depenent de la direcció del paquet i s'assignen els diversos flags del protocol [79]:

Codi 71 Capturer.java: funció vbsPacket() - encapsulament TCP

```
...
//TCP value: Transmission Control (TCP): 6
if (p.equals("6")) {
   TcpHeader tcpHeader = packetIPv4.get(TcpPacket.class).getHeader();
   newPacket.setProtocolName("TCP");
   srcPort = tcpHeader.getSrcPort().valueAsInt();
   dstPort = tcpHeader.getDstPort().valueAsInt();
   if (newPacket.getPacketDirection() == 'O') {
      newPacket.setLocalPort(srcPort);
      newPacket.setRemotePort(dstPort);
   } else {
      newPacket.setLocalPort(dstPort);
      newPacket.setRemotePort(srcPort);
   }
    newPacket.setFlagACK(tcpHeader.getAck());
    newPacket.setFlagFIN(tcpHeader.getFin());
    newPacket.setFlagPSH(tcpHeader.getPsh());
    newPacket.setFlagRST(tcpHeader.getRst());
    newPacket.setFlagSYN(tcpHeader.getSyn());
    newPacket.setFlagURG(tcpHeader.getUrg());
}
...
```

Si el paquet és *ICMP* només s'assigna el nom del protocol i es defineixen els ports a zero, ja que no és de la capa de transport:

Codi 72 Capturer.java: funció vbsPacket() - Encapsulament ICMP

```
...
//ICMPv4: Internet Control Message (ICMPv4): 1
else if (p.equals("1")) {
   newPacket.setProtocolName("ICMP");
   newPacket.setLocalPort(0);
   newPacket.setRemotePort(0);
}
...
```

Per últim, en el protocol *UDP* s'obtenen els ports i s'assignen a local o remot depenent de la direcció del paquet [80]:

Codi 73 Capturer.java: funció vbsPacket() - Encapsulament UDP

```
...
else if (p.equals("17")) {
   newPacket.setProtocolName("UDP");
   UdpHeader udpHeader = packetIPv4.get(UdpPacket.class).getHeader();
   srcPort = udpHeader.getSrcPort().valueAsInt();
   dstPort = udpHeader.getDstPort().valueAsInt();
   if (newPacket.getPacketDirection() == 'O') {
      newPacket.setLocalPort(srcPort);
      newPacket.setRemotePort(dstPort);
   }
   else {
      newPacket.setLocalPort(dstPort);
      newPacket.setRemotePort(srcPort);
   }
}
...
```

Si s'ha pogut extreure tots els camps del paquet rebut i s'ha creat correctament el paquet VBS, es considera que aquest és vàlid. Així que s'inclou a l'estructura *myPacketQueue* perquè la classe *Flow Generator* el pugui incloure a la traça corresponent:

Codi 74 Capturer.java: funció vbsPacket() - Classe CapturedPacketsQueue

```
/** Queue of captured packets. */
private CapturedPacketsQueue myPacketQueue = null;

...
myPacketQueue.queueUpPacket(newPacket);
return true;
...
```

Mode Debug

Durant l'implementació d'aquesta classe s'han afegit algunes funcionalitats de cara a debugar el codi. Aquestes funcions s'han deixat comentades i no s'han eliminat per si en algun futur es vol realitzar una apliació del software.

S'ha implementat la funció *printVbsPacket* perquè no es va trobar cap manera de saber quin era el resultat del paquet VBS creat. És un codi molt senzill que obté tots els atributs del paquet VBS i els afageix al logger:

Codi 75 Capturer.java: funció printVbsPacket()

```
private void printVbsPacket(CapturedPacket newPacket) {
    log.debug("New Packet: " + newPacket);
    log.info("Local IP: " + newPacket.getLocalIP());
    log.info("Dest IP: " + newPacket.getRemoteIP());
    log.debug("Local Port: " + newPacket.getLocalPort());
    log.debug("Dest Port: " + newPacket.getRemotePort());
    log.info("Protocol: " + newPacket.getProtocolName());
    log.debug("Content Type (HTTP): " + newPacket.getContentType());
    log.debug("URL (HTTP): " + newPacket.getUrl());
    log.debug("Refer (HTTP): " + newPacket.getReferer());
    log.info("Packet Direction: " + newPacket.getPacketDirection());
    log.debug("Packet size: " + newPacket.getPacketSize());
    log.info("Get Timestamp: " + newPacket.getTimestamp());
    log.debug("ACK Flag: " + newPacket.getFlagACK());
    log.debug("CWR Flag: " + newPacket.getFlagCWR());
    log.debug("ECN Flag: " + newPacket.getFlagECN());
    log.debug("FIN Flag: " + newPacket.getFlagFIN());
    log.debug("PSH Flag: " + newPacket.getFlagPSH());
    log.debug("RST Flag: " + newPacket.getFlagRST());
    log.debug("SYN Flag: " + newPacket.getFlagSYN());
    log.debug("URG Flag: " + newPacket.getFlagURG());
    log.debug("Payload: " + newPacket.getPayload());
}
```

Durant les proves per capturar paquets IPv4, també se'n van realitzar amb altres tipus de paquets. S'ha configurat un altre handler perquè envii paquets al client i s'ha creat un paquet amb l'encapsulament *ARP*[81] i *Ethernet* [82] a mà:

Codi 76 Capturer.java: funció run() - Paquet ARP, Ethernet i sendHandle

```
PcapHandle sendHandle;
sendHandle = nif.openLive(SNAPLEN, PromiscuousMode.PROMISCUOUS,
    READ_TIMEOUT);
/* Arp Builder */
ArpPacket.Builder arpBuilder = new ArpPacket.Builder();
try {
   arpBuilder
    .hardwareType(ArpHardwareType.ETHERNET)
    .protocolType(EtherType.IPV4)
    .hardwareAddrLength((byte)MacAddress.SIZE_IN_BYTES)
        .protocolAddrLength((byte)ByteArrays.INET4_ADDRESS_SIZE_IN_BYTES)
    .operation(ArpOperation.REQUEST)
    .srcHardwareAddr(SRC_MAC_ADDR)
    .srcProtocolAddr(InetAddress.getByName(FormatConverter.decimalFormat(IPAddress)))
    .dstHardwareAddr(MacAddress.ETHER_BROADCAST_ADDRESS)
    .dstProtocolAddr(InetAddress.getByName(FormatConverter.decimalFormat(IPAddress)));
} catch (UnknownHostException e) {
   throw new IllegalArgumentException(e);
}

/* Ethernet Builder */
EthernetPacket.Builder etherBuilder = new EthernetPacket.Builder();
etherBuilder.dstAddr(MacAddress.ETHER_BROADCAST_ADDRESS)
   .srcAddr(SRC_MAC_ADDR)
   .type(EtherType.ARP)
   .payloadBuilder(arpBuilder)
   .paddingAtBuild(true);

for (int i = 0; i < COUNT; i++) {
   Packet p = etherBuilder.build();
try {
   sendHandle.sendPacket(p);
} catch (Exception e) {
   e.printStackTrace();
}
try {
   Thread.sleep(1000);
} catch (InterruptedException e) {
```

```
   break;
}
}
```

Aquesta estructura de creació del paquet dels builders existeix amb tots els protocols. Senzillament consisteix en començar creant el builder del protocol de la capa més elevada. Es defineixen tots els atributs i es passa el builder al del següent encapsulament. Finalment es crea el paquet usant un bucle s'envien. Definint la variable *COUNT* es pot determinar la quantitat de paquets a enviar.

6.4.4 Classe SocketInfoGenerator

Els principals canvis que s'han hagut d'implementar en aquesta classe estan directament relacionats en l'obtenció de l'informació dels sockets, ja que la tècnica usada a Linux no és comptetible amb Mac OS X.

Funció setOperatingSystem

Degut a que aquetes tècniques varien entre els diferents sistemes operatius, el primer que s'ha de fer és detectar quin té el client. Per fer-ho, es declara una variable global per cada sistema operatiu i a través de la següent funció, es determina quin és:

Codi 77 SocketInfoGenerator.java: funció setOperatingSystem

```
public final static String OSNAMEOSX = "mac os x";
...
private static void setOperatingSystem() {
   String nameOS = "os.name";
   String OSname = System.getProperty(nameOS); // Windows 7
   OSname = OSname.toLowerCase();
   if (OSname.contains(OSNAMEWINDOWS)) {
      operatingSystem = OSNAMEWINDOWS;
   } else if (OSname.contains(OSNAMELINUX)) {
      operatingSystem = OSNAMELINUX;
   } else if (OSname.contains(OSNAMEOSX)) {
      operatingSystem = OSNAMEOSX;
   }else
      log.error("Could not identify operating system " + OSname);
}
```

Funció getNestatOutput

Per obtenir l'informació dels diferents sockets s'executen les següents comandes:

- *netstat –inet -anp* a Linux [29].
- *tcpvcon.exe -acn* a Windows [30].
- *lsof -Pnl -i4* a Mac OS X [31].

Tot i que *netstat* també existeix en els sitemes Mac OS X, s'ha decidit usar lsof degut a que usant la primera no es pot obtenir la mateixa informació, ja que no mostra el nom del socket en qüestió. És per això que aquesta funció s'ha hagut de modificar lleugerament afegint la comanda adequada:

Codi 78 SocketInfoGenerator.java: funció getNestatOutput

```
private final static String[] netStatMac = {"lsof", "-Pnl", "-i4"};
...
private String[] getNestatOutput() {
...
    String[] cmd_elements = null;
    if (getOperatingSystem().equals(OSNAMEWINDOWS)) {
        cmd_elements = netStatWindows;
    } else if (getOperatingSystem().equals(OSNAMELINUX)) {
        cmd_elements = netStatLinux;
    } else if (getOperatingSystem().equals(OSNAMEOSX)){
        cmd_elements = netStatMac;
    }
...
}
```

Funció netstatLineTokenizer

A nivell estructural, aquesta classe primer obté el resultat de l'execució de la comanda, després crida a aquesta funció i per últim, comprova que cada camp obtingut és el correcte i els assigna als sockets corresponents.

Per a poder accedir als valors obtinguts després de l'execució, aquests s'han de tractar abans. Aquesta fucnió rep el resultat en format string i, en Linux i Mac OS X, posa a la variable l'array *token* cada valor en una posició diferent:

Codi 79 SocketInfoGenerator.java: funció netstatLineTokenizer - Creació token

```
private String[] netstatLineTokenizer(String stringToTokenize) {
...
   String[] token;
   if (getOperatingSystem().equals(OSNAMELINUX) ||
       getOperatingSystem().equals(OSNAMEOSX)) {
      StringTokenizer stringTokenizer = new
          StringTokenizer(stringToTokenize);
      token = new String[stringTokenizer.countTokens()];
      for (int i = 0; stringTokenizer.hasMoreTokens(); i++) {
         token[i] = stringTokenizer.nextToken();
      }
   }
...
}
```

En sistemes Linux, no cal realitzar cap altra modificació ja que tota la classe està programada al voltant d'aquest sistema operatiu. No obstant això, en el nostre cas s'ha hagut d'implementar un petit parser dins d'aquesta funció:

Codi 80 SocketInfoGenerator.java: funció netstatLineTokenizer - Mac OS X parser

```
private String[] netstatLineTokenizer(String stringToTokenize) {
...
   if (getOperatingSystem().equals(OSNAMEOSX)) {
      String[] tokenAux = new String[7];
      tokenAux[0] = token[7];
      tokenAux[1] = "";
      tokenAux[2] = "";
      tokenAux[6] = token[0];
      if (token.length == 10) {
      try {
         tokenAux[3] = token[8].split("->")[0];
         tokenAux[4] = token[8].split("->")[1];
      } catch (Exception e) {
         tokenAux[3] = "";
         tokenAux[4] = "";
      }
         tokenAux[5] = token[9];
```

```
      }
      else if (token.length == 9) {
         if (token[8].contains("->")) {
            tokenAux[3] = token[8].split("->")[0];
            tokenAux[4] = token[8].split("->")[1];
         } else {
            tokenAux[3] = token[8].split(":")[0];
            tokenAux[4] = token[8].split(":")[1];
         }
         tokenAux[5] = "";
      }
      token = new String[7];
      token = tokenAux;
   }
return token;
}
```

Principalment, es crea una variable anomenada *tokenAux* que serà on es guardaran les dades després del parseig.

El resultat de l'execució de la comanda *lsof* el rebem així despés de posar-lo a l'array (exemple socket de Google Chrome):

- Posició 0 (nom): Google
- Posició 1 (PID): 19154
- Posició 2 (User): 501
- Posició 3 (FD): 99u
- Posició 4 (Tipus): IPv4
- Posició 5 (Dispositiu): 0x52ffa99ec69a34c9
- Posició 6 (Mida): 0t0
- Posició 7 (Node): TCP
- Posició 8 (IP/Port): 192.168.1.45:59758->216.58.214.161:443
- Posició 9 (Estat): (ESTABLISHED)

Totes les posicions, excepte la vuitena, no necessiten cap mena de parseig. Només cal ordenar-les. En canvi, la posició 8 s'ha de tractar. En el nostre cas el que es fa és un split de l'string usant les caràcters "->" com a punt, per obtenir les adreces i ports locals remots.

Tot i així, s'ha de tenir en compte també el cas on el socket no tingui una adreça o port determinat. Ens hem trobat exemples on en comptes s'una IP o port hi ha el caràcter "*". Per tractar aquests casos, s'ha afegit una condició extra.

Tenint en compte l'exemple, després d'executar el codi la funció retornaria el següent array:

- Posició 0 (Node): TCP
- Posició 1 (Bytes in): 0
- Posició 2 (Bytes out): 0
- Posició 3 (IP/Port Local): 192.168.1.45:59758
- Posició 4 (IP/Port Remot): 216.58.214.161:443
- Posició 5 (Estat): (ESTABLISHED)
- Posició 6 (Nom): Google

Si després d'executar la funció *findSocketInfo* es verifica que tots els camps obtinguts són correctes, s'afegirà el socket a un array que usarà la funció *Flow Generator* per linkar-lo amb la traça corresponent.

6.5 Base de dades

En aquest apartat es mostra com s'emmagatzemen els paquets i traces a la base de dades del client i del servidor. D'aquesta manera es poden observar com es tracten i de quina manera estan relacionades les diferents dades recollides durant l'execució.

6.5.1 Client

La base de dades *SQLite* del client està formada per tres taules. La primera de totes és la taula anomenada "Client" i és la primera que s'omple. Aquesta conté informació relacionada amb el client: identificació, IP global, informació sobre el sitema operatiu i la versió del software VBS que està executant el client:

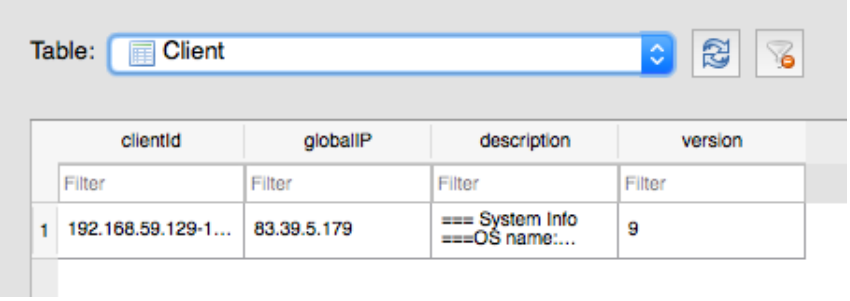
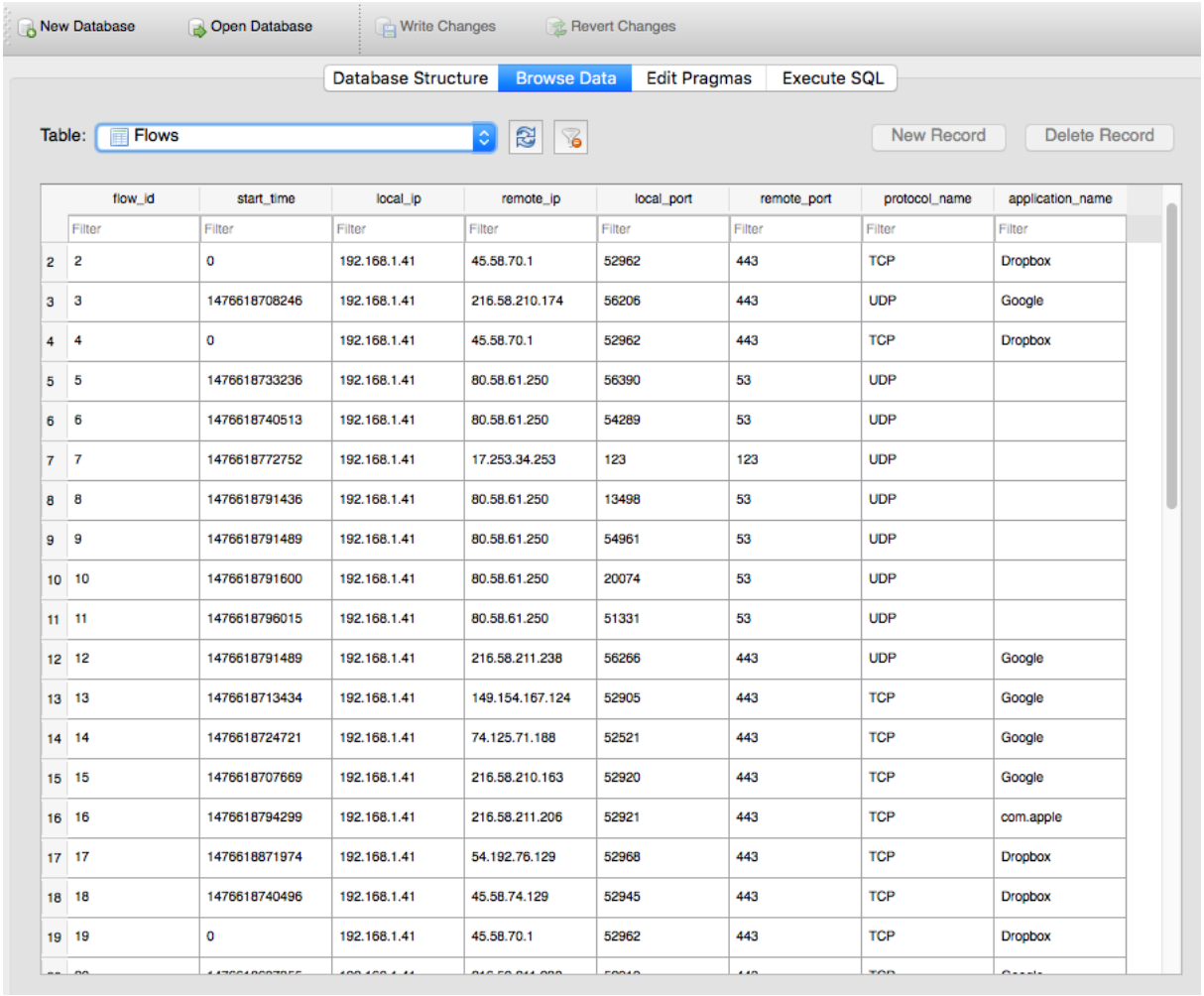


Figura 3 Base de dades client - Taula Client

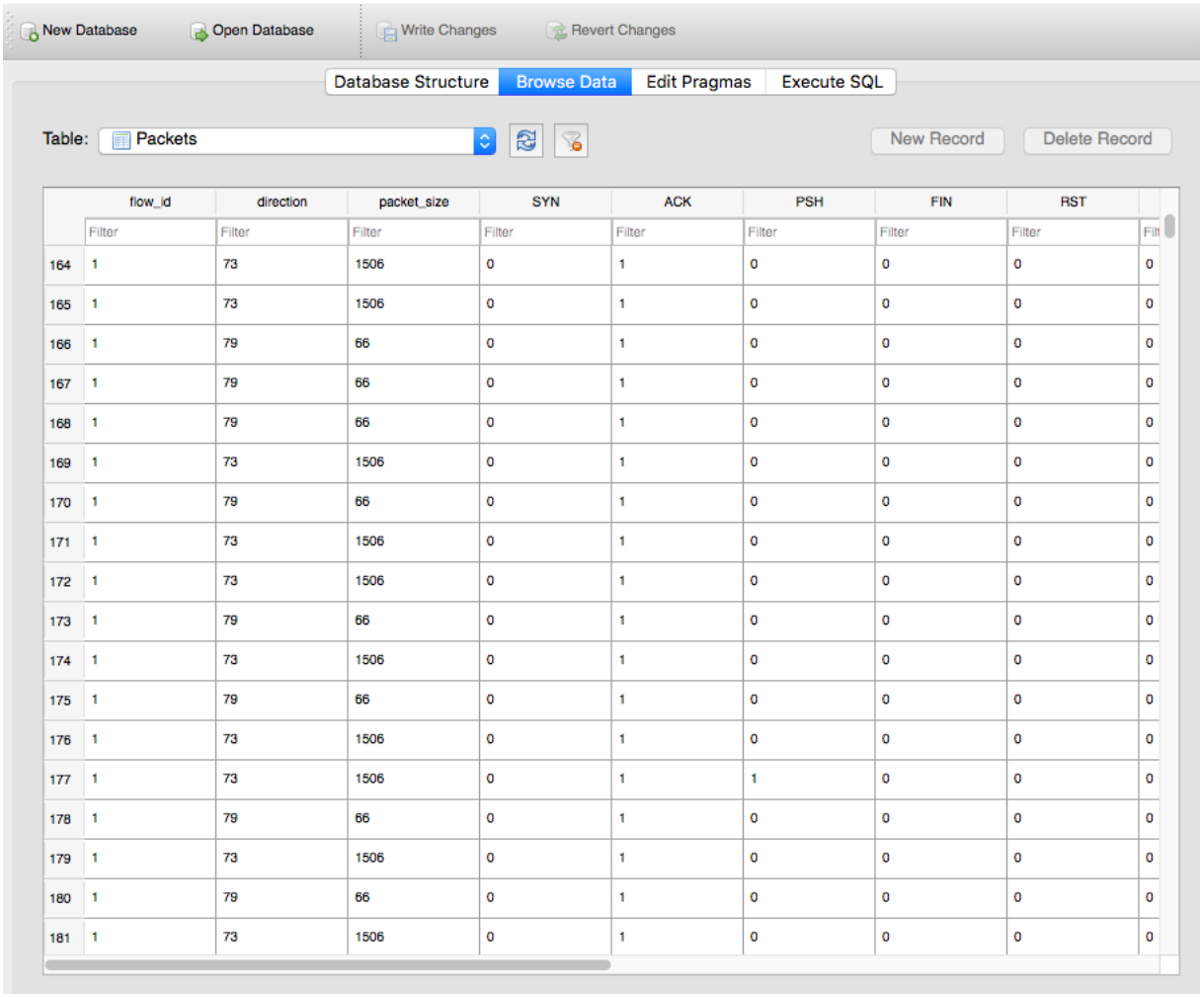
La taula anomenada "Flows" és on s'emmagatzemen totes les traces creades durant l'execució del programa. Cada traça consta d'un identificador, camp de relació amb els paquets, el temps d'inici, l'IP Origen que acostuma a ser la del client degut a que és qui comença la conversa, l'IP destí, port local, port remot, protocol usat i el nom de l'aplicació encarregada de generar els paquets pertinyents a la traça:



	flow_id	start_time	local_ip	remote_ip	local_port	remote_port	protocol_name	application_name
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
2	2	0	192.168.1.41	45.58.70.1	52962	443	TCP	Dropbox
3	3	1476618708246	192.168.1.41	216.58.210.174	56206	443	UDP	Google
4	4	0	192.168.1.41	45.58.70.1	52962	443	TCP	Dropbox
5	5	1476618733236	192.168.1.41	80.58.61.250	56390	53	UDP	
6	6	1476618740513	192.168.1.41	80.58.61.250	54289	53	UDP	
7	7	1476618772752	192.168.1.41	17.253.34.253	123	123	UDP	
8	8	1476618791436	192.168.1.41	80.58.61.250	13498	53	UDP	
9	9	1476618791489	192.168.1.41	80.58.61.250	54961	53	UDP	
10	10	1476618791600	192.168.1.41	80.58.61.250	20074	53	UDP	
11	11	1476618796015	192.168.1.41	80.58.61.250	51331	53	UDP	
12	12	1476618791489	192.168.1.41	216.58.211.238	56266	443	UDP	Google
13	13	1476618713434	192.168.1.41	149.154.167.124	52905	443	TCP	Google
14	14	1476618724721	192.168.1.41	74.125.71.188	52521	443	TCP	Google
15	15	1476618707669	192.168.1.41	216.58.210.163	52920	443	TCP	Google
16	16	1476618794299	192.168.1.41	216.58.211.206	52921	443	TCP	com.apple
17	17	1476618871974	192.168.1.41	54.192.76.129	52968	443	TCP	Dropbox
18	18	1476618740496	192.168.1.41	45.58.74.129	52945	443	TCP	Dropbox
19	19	0	192.168.1.41	45.58.70.1	52962	443	TCP	Dropbox

Figura 4 Base de dades client - Taula Flows

Per últim, tots els paquets capturats es guarden a la taula "Packets". Aquesta conté tots els camps de la classe VBSPacket: direcció, flags, IP origen, IP destí... El camp usat per relacionar el paquet amb la seva traça és el flow_id.



	flow_id	direction	packet_size	SYN	ACK	PSH	FIN	RST	
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filt
164	1	73	1506	0	1	0	0	0	0
165	1	73	1506	0	1	0	0	0	0
166	1	79	66	0	1	0	0	0	0
167	1	79	66	0	1	0	0	0	0
168	1	79	66	0	1	0	0	0	0
169	1	73	1506	0	1	0	0	0	0
170	1	79	66	0	1	0	0	0	0
171	1	73	1506	0	1	0	0	0	0
172	1	73	1506	0	1	0	0	0	0
173	1	79	66	0	1	0	0	0	0
174	1	73	1506	0	1	0	0	0	0
175	1	79	66	0	1	0	0	0	0
176	1	73	1506	0	1	0	0	0	0
177	1	73	1506	0	1	1	0	0	0
178	1	79	66	0	1	0	0	0	0
179	1	73	1506	0	1	0	0	0	0
180	1	79	66	0	1	0	0	0	0
181	1	73	1506	0	1	0	0	0	0

Figura 5 Base de dades client - Taula Packets

6.5.2 Servidor

Quan el servidor rep la còpia de la base de dades SQLite, l'importa al MySQL. Les taules són molt similars a les del client. Per emmagatzemar tota l'informació dels diferents clients que estan enviant informació al servidor, s'usa la taula "Clients". Aquesta consta de tres camps: l'id del client assignat pel servidor, l'identificador i la descripció del client (sistema operatiu, CPU, arquitectura...):

#	client_id	clientId	description
	1	192.168.59.129-1467731710006	=== System Info === OS name: Mac OS X OS version: 10.11.5 OS architecture: x86_64 CPU[0] information: {TotalSockets=4, CoresPerSocket=16, CacheSize=256, Model=MacBook… CPU[1] information: {TotalSockets=4, Cor...
	NULL	NULL	NULL

Figura 6 Base de dades servidor - Taula Clients

Per emmagatzemar les traces, s'usa una taula gaire bé idèntica amb la del client. L'única diferència és que en comptes de tenir una columna amb el nom de l'aplicació que ha generat els paquets, té un identificador que el relaciona amb una altra taula que conté els noms:

#	flow_id	client_id	start_time	local_ip	remote_ip	local_port	remote_port	protocol_name	global_local_ip	application_id
86	86	1	1476618358842	192.168.1.41	147.83.2.135	52844	80	TCP	83.39.5.179	2
87	87	1	1476618362000	192.168.1.41	54.89.42.185	52525	443	TCP	83.39.5.179	2
88	88	1	1476618359881	192.168.1.41	54.231.13...	52857	443	TCP	83.39.5.179	2
89	89	1	1476618358858	192.168.1.41	147.83.41.15	52851	443	TCP	83.39.5.179	2
90	90	1	1476618405489	192.168.1.41	147.83.2.135	52883	80	TCP	83.39.5.179	3
91	91	1	1476618559617	192.168.1.41	80.58.61.250	65416	53	UDP	83.39.5.179	1
92	92	1	1476618542363	192.168.1.41	17.253.37....	52918	80	TCP	83.39.5.179	4
93	93	1	1476618358846	192.168.1.41	147.83.2.135	52845	80	TCP	83.39.5.179	2
94	94	1	1476618361491	192.168.1.41	52.48.68.196	52858	443	TCP	83.39.5.179	2
95	95	1	1476618348008	192.168.1.41	216.58.21...	52838	443	TCP	83.39.5.179	2
96	96	1	1476618589154	192.168.1.41	17.154.66....	52939	443	TCP	83.39.5.179	4
97	97	1	1476618439564	192.168.1.41	17.252.27....	52891	443	TCP	83.39.5.179	8
98	98	1	1476618333909	192.168.1.41	54.85.157.47	52523	443	TCP	83.39.5.179	2

Figura 7 Base de dades servidor - Taula Flows

La taula que conté els noms de les aplicacions s'anomena "Applications". Es va decidir fer aquesta taula intermitja per a poder centralitzar totes les aplicacions de les traces de tots els clients. Els únics camps que té són un identificador i el nom:

#	application_id	application_name
1	1	
2	11	Appx20St
3	12	Dashboard
4	6	Dropbox
5	2	Google
6	9	Safari
7	8	Spotlight
8	5	akd
9	10	apsd
10	3	com.apple
11	7	storeacco
12	4	storedown

Figura 8 Base de dades servidor - Taula Flows

Per últim, hi ha la taula dels paquets. Aquesta és exactament igual que la del client exceptuant la columna "content_type_id" que correspon a una taula anomenada "ContentTypes" que en el nostre cas està buida, ja que conté informació dels paquets HTTP, igual que els camps "refer" i "url".

#	packet_id	flow_id	direction	packet_size	SYN	ACK	PSH	FIN	RST	CWR	ECN	URG	real_timestamp	content_type_id	referer	url	payload
26	26	12	O	83	0	0	0	0	0	0	0	0	1476618346251	1			BLOB
27	27	12	I	118	0	0	0	0	0	0	0	0	1476618346255	1			BLOB
28	28	13	O	84	0	0	0	0	0	0	0	0	1476618346642	1			BLOB
29	29	13	I	144	0	0	0	0	0	0	0	0	1476618346655	1			BLOB
30	30	14	O	84	0	0	0	0	0	0	0	0	1476618346643	1			BLOB
31	31	14	I	144	0	0	0	0	0	0	0	0	1476618346659	1			BLOB
32	32	15	O	78	1	0	0	0	0	0	0	0	1476618346643	1			BLOB
33	33	15	I	74	1	1	0	0	0	0	0	0	1476618346686	1			BLOB
34	34	15	O	66	0	1	0	0	0	0	0	0	1476618346686	1			BLOB
35	35	15	O	543	0	1	1	0	0	0	0	0	1476618346686	1			BLOB
36	36	15	I	66	0	1	0	0	0	0	0	0	1476618346714	1			BLOB
37	37	15	I	74	0	1	1	0	0	0	0	0	1476618346725	1			BLOB
38	38	15	O	78	0	1	0	0	0	0	0	0	1476618346725	1			BLOB

Figura 9 Base de dades servidor - Taula Paquets

6.6 Comparativa eines DPI

Per acabar, s'ha creat una petita base de dades amb tràfic generat des d'un ordinador amb *Mac OS X* usant les següents aplicacions:

- Navegador Google Chrome
- Navegador Safari
- Dropbox
- Spotlight
- iMessages
- AppStore
- Telegram

S'ha obtingut l'arxiu PCAP amb 16232 paquets, s'han usat dues eines *DPI*, *nDPI* i *libprotoident*, perquè l'analitzessin i s'han comparat els resultats.

S'ha observat que *Apple* no sempre assigna com a nom del procés el nom de l'aplicació. Això provoca que tot i executar la comanda *lsof* per obtenir els noms, hi ha algunes aplicacions que tenen el mateix. Per exemple, el nom obtingut de les aplicacions que hem executat són els següents:

- **Navegador Google Chrome**: Google
- **Navegador Safari**: Safari
- **Dropbox**: Dropbox
- **Spotlight**: Spotlight
- **iMessages**: com.apple
- **AppStore**: com.apple
- **Telegram**: com.apple

Es pot observar que les últimes tres aplicacions tenen el mateix nom. Això provoca que les *eines DPI* no puguin realtizar l'anàlisi correctament. S'han analitzat els resultats d'ambudes eines i s'han extret les següents conclusions:

Procés	nDPI	libprotoident
HTTP	5140	8196
HTTPS	-	4755
Chrome	2682	-
SSL	3130	2169
Dropbox	191	-
Apple	4783	-
Altres	289	627
No reconeguts	80	485

Taula 2 Comparativa eines DPI

Els resultats obtinguts no han sigut del tot concluents degut a que les eines no han detectat del tot bé les diferents traces pel seu nom del procés. No han pogut separar els paquets de *com.apple* i tampoc han pogut determinar els paquets del programa *Spotlight* ni *Safari*, els qual segurament s'han classificat com a *HTTP* i *HTTPS*.

Si es comparen les dues eines, *nDPI* té més precissió que *libprotoident*. Per una banda, *nDPI* té menys paquets no reconeguts i per l'altra, ha sigut capaç de mostrar una resultats més acurats a l'hora d'indicar d'on provenen els paquets, ja que ha pogut determinar quins són de *Chrome*, quins d'*Apple* i quins de *Dropbox*.

Si s'intenta acotar el resultat, es podria realitzar la suposició de que part dels paquets *HTTPS* pertanyen segurament al navegador *Safari* perquè al generar el tràfic s'ha accedit a una web *HTTP* i a una altra *HTTPS*. També podria suposar que els paquets *HTTPS* de l'eina libprotoident són els mateixos que els etiquetats com a *Apple* per *nDPI* i per tant, podrien pertànyer al *Safari* juntament amb part dels paquets *HTTP*. Per últim, els paquets *SSL* podrien pertànyer a les applicacions de missatgeria, *Telegram* i *iMessages* [83], i a l'*AppStore* [84] degut a que les tres usen el protocol *SSL/TLS* [85] per a realitzar comunicacions segures.

7. Gestió econòmica

7.1 Consideracions inicials

En els següents apartats s'analitzaran els costos derivats al desenvolupament del projecte. Com que es tracta d'un projecte universitari, les estimacions s'han realitzat tenint en compte els costos associats a cada una de les tasques i el rol del desenvolupador en condició de becari.

7.2 Identificació i estimació de costos

La gestió econòmica ha estat dividida en costos directes (recursos humans, hardware i software), on s'ha tingut en compte la vida útil del producte per saber l'amortització, i costos indirectes (llum i Internet) on s'ha calculat el que es pot gastar en els 5 mesos que dura el projecte. A més a més, s'ha considerat les possibles desviacions del pressupost, com es detectaran i quin impacte tindran en el projecte.

7.2.1 Costos directes

Recursos humans

Els recursos humans del projecte només els constitueixen un estudiant d'enginyeria informàtica especialitzat en tecnologies de la informació. Al no tenir el títol, s'ha considerat que cobra com un becari de la FIB: 7.5€ l'hora.

Tasca	Estimació de temps	Cost
Gestió de projecte	76 hores	570€
Adaptació a l'entorn	44 hores	330€
Configuració de l'entorn	40 hores	300€
Identificació i solució de problemes de compatibilitat	60 hores	450€
Desenvolupament	280 hores	2100€
Etapa final	100 hores	750€
Total	**600 hores**	**4500€**

Taula 3 Costos directes: Recursos humans

Hardware

S'usarà un ordinador de sobretaula des d'on s'executaran les diverses màquines virtuals. A més a més, també es treballarà des d'un portàtil amb connexió remota per quan s'hagi de treballar des de l'universitat o biblioteca.

Producte	Cost	Vida útil	Amortització total estimada (5 mesos)
Ordinador de sobretaula	1500€	5 anys	92€
Macbook Air	950€	5 anys	58€
Total	150€		

Taula 4 Costos directes: Hardware

Software

El software usat en aquest projecte és de cost 0€, ja que software té llicència de software lliure o són llicències d'ús gratuït. No obstant això hi ha dos programes que no compleixen aquestes condicions:

- VMWare: és software privatiu que s'ha de pagar sempre i quan sigui d'ús comercial. Com que aquest projecte és per l'universitat i en cap moment es vol comercialitzar, és gratuït.
- Windows 7: aquest sistema operatiu es pot trobar entre els 30€ i 70€. Al ser estudiant de l'UPC, l'universitat posa a la disposició dels alumnes una llicència gratuïta.

7.2.2 Costos indirectes

Els costos indirectes associats al projecte són complicats de calcular ja que hi ha molta informació que no és de fàcil accés com l'aigua i el consum elèctric. Per això, només es tindrà l'accés a Internet i transport (T-Jove) amb valors aproximats.

Producte/Servei	**Cost**	**Període**	**Amortització total estimada (5 mesos)**
Accés a Internet	15€	5 mesos	60€
Transport	100€/bitllet	5 mesos	166€
Total	226€		

Taula 5 Costos indirectes

7.2.3 Contingència

Un altre especte a tenir en compte en el pressupost és la partida de contingència. En aquest projecte es destinarà a aquesta un 15% de la suma dels costos directes i indirectes.

Concepte	**Procentatge**	**Cost**	**Total)**
Costos directes	15%	4650€	697.5€
Costos indirectes	15%	226€	33.9€
Total	731.4€		

Taula 6 Costos de contingència

7.2.4 Imprevistos

Tal i com s'ha mencinat a les seccions anteriors, la planificació temporal s'ha realitzat tenint en compte una sèrie d'imprevistos:

- **Retràs de 7 dies:** ja que la part més sensible de sofrir una desviació temporal és la de l'implementació, és la més imprevisible. Es suposarà una jornada de 8 hores.
- **Avaria del hardware:** si algun dels ordinadors es fes malbé, s'ha de tenir en compte el cost de reparació. S'agafarà un cost de reparació intermig ja que és gairebé impossible que els ordinadors deixin de funcionar completament.

Concepte	Probabilitat	Cost
Retràs de 7 dies	20%	420€
Reparació ordinador sobretaula	5%	50€
Reparació MacBook Air	5%	100€
Total	570€	

Taula 7 Pressupost per imprevistos

7.2.5 Pressupost

En aquest projecte no s'ha tingut en compte els beneficis que es podrien generar a través de la comercialització d'aquest software, ja que és de caràcter universitari. A més a més, no es contempla una possible pujada del salari durant el desenvolupament del projecte.

Concepte	Cost
Costos directes	4650€
Costos indirectes	226€
Contingència	731.4
Imprevistos	570€
Total	**6117.4€**

Taula 8 Pressupost del projecte

7.3 Control de gestió

Amb la finalitat de controlar l'evolució del projecte es realitzarà una imputació d'hores al final de la setmana. D'aquesta manera, es sabrà quantes hores queden disponibles per a realitzar les tasques.

A través d'aquest control és possible detectar si les hores dedicades a cada tasca sobrepassen el pressupost estimat o no. En cas afirmatiu, s'identificaran les causes que han produït aquest desajustament pressupostari i es modificaran les següents tasques per reduïr la diferència de costos.

Si finalment, el cost real supera el pressupostat, s'aplicaria la partida d'imprevists i contingència per cobrir la diferència.

8. Sostenibilitat i compromís social

8.1 Valoració de la sostenibilitat

Per a poder realitzar correctament un estudi de sostenibilitat, es valora des de tres punts de vista: l'econòmic, el social i l'ambiental. Cada una d'aquestes dimensions obté una puntuació que es calcula usant el mètode socràtic. És a dir, contestant una sèrie de preguntes i assignant una puntuació objectiva a cada una d'elles.

L'estudi es pot resumir amb la següent taula:

Sostenibilitat	**Econòmica**	**Social**	**Ambiental**	**Total**
Planificació	Viabilitat econòmica	Millora de la qualitat de vida	Anàlizi de recursos	
Valoració	8.3	7.1	8.6	8

Taula 9 Puntuacions totals de l'estudi de sostenibiltat

L'alta puntuació obtinguda, 8 punts, fa que aquest projecte sigui viable per ser desenvolupat ja que té en general un impacte positiu en els tres àmbits que es detallen a continuació.

8.2 Econòmica

La dimensió econòmica té una puntuació de 8.3:

- En aquest projecte es realitza una avaluació dels recursos materials i humans tal i com reflecteix l'apartat anterior.
- Aquest projecte requereix un mínim manteniment ja que els sistemes operatius es van actualtizant any rere any. No obstant això, aquest és casi nul.
- El projecte seria viable ja que la part més gran del pressupost és per a pagar els desenvolupadors. Les despeses en hardware i software són ínfimes i per tant podria ser molt competitiu.

- Es podria realitzar el projecte per un cost menor si l'enginyer encarregat el pogués fer en menys hores o aquestes fossin més barates.
- Com més important és la part del projecte, més hores s'hi dediquen.

8.3 Social

La dimensió social té una puntuació de 7.1:

- La situació social del país és benestant, tot i que la política està patint molt canvis.
- El sector que inclou aquest projecte no té un impacte directe en la població però indirectament si que ajuda al progrés i a la generació de riquesa, ja que pot afavorir a algunes empreses.
- Aquest és un projecte que no té un impacte directe en la situació que viu el país actualment.
- Existeix una necessitat ja que no existeix cap manera per a poder valorar i comparar les eines de Deep Package Inspection en l'actualitat.
- Als clients que vulguin els serveis d'alguna d'aquestes eines els afavorirà aquest projecte ja que tindran una manera de poder comparar-los i escollir el que més s'adapti a les seves necessitats.
- Pels usuaris aquest projecte no aporta res, però si a les empreses que usin eines de Deep Package Inspection, ja que poden saber quins són els seus punts forts i febles.
- Aquest projecte no afecta a cap col·lectiu negativament.

8.4 Ambiental

La dimensió social té una puntuació de 8.6:

- L'únic recurs que pot afectar al medi ambient és la llum. Depén de com hagi estat produïda: si de fonts renovables o no renovables.
- Durant la fase de desenvolupament l'impacte ambiental és ínfim ja que només requereix l'ús d'un parell d'ordinadors. I durant la seva vida útil, directament no consumirà res ja que són les pròpies empreses les que usaran els resultats a través de les seves infraestructures.

- Les empreses podran valorar els resultats i pensar si l'eina d'anàlisi de paquets que estan usant actualment és suficientment eficient per a complir les seves necessitats o si haurien de canviar. Això pot produir un decrement del consum si, per exemple, una empresa decideix usar una eina d'anàlisi de paquets que tarda un 5% menys de temps a analitzar un cert tipus de traces.
- Es poden reutilitzar moltes GB de traces creades en un projecte anterior. Això reduirà el temps d'ús dels ordinadors i per tant, el consum.
- L'energia usada és molt poca, només la necessària per a poder crear el tràfic i capturar les traces.
- No es realitzarà desmantellament perquè no és necessari.
- Només produirà contaminació indirectament.
- No requereix material manufacturat.
- No tenen cap implicació ètica ja que tot el tràfic capturat és simulat i per tant, no es compromet la privacitat de ningú.
- Segurament no tindrà un impacte tan gran com per disminuir l'empremta ecològica, però segur que no l'augmentarà.
- Es pot reciclar tot el tràfic i traces produïdes, ja que estan dividides per tipus i poden ser usades en diferents projectes i sistemes operatius.

9. Conclusions

Actualment estem a l'era de la informació i el món ha esdevingut completament dependent de la tecnologia. Gairebé qualsevol pot aconseguir accés a Internet amb molt pocs recursos. És més, es podria dir que persones i empreses estan connectades a Internet 24 hores al dia.

Això ha provocat que moltes companyies es preocupin cada vegada més per la ciber seguretat. No només inverteixen grans sumes en formació, antivirus i anàlisis periòdics, sinó que també n'inverteixen en detectar amenaces abans que aquestes arribin als ordinadors interns.

En aquest punt entra en joc el software conegut com a *Deep Package Inspection Tools* que permet analitzar el tràfic de paquets per detectar possible contingut maliciós.

Per a poder millorar i evaluar aquestes eines, és necessari realitzar estudis per saber quin tipus de tràfic detecten correctament i quin no. Un software que permet realitzar aquest estudi és l'anomenat *Volunteer-Based System (VBS)*.

Aquest projecte s'ha centrat en millorar el *software VBS*. Per una banda, ha demostrat que no és possible una adaptació de la llibreria usada per capturar i classificar paquets *jNetPcap* perquè funcioni en sistemes *Mac OS X*, ja que no s'ha pogut crear la llibreria en format .dylib sense haver de programar-la des de zero. Durant aquest intent de compatibilització, s'ha investigat a fons com funcionen i quines són les diferències entre les llibreries de *Linux* i *Mac OS X*. També s'ha descobert i usat *JNI* per intentar compilar la llibreria en el format correcte ja que ofereix la funcionalitat de comunicar classes programades en diferents llenguatges de programació, *Java* i *C* en el nostre cas.

Per altra banda, s'ha reimplementat el *software VBS* perquè sigui compatible amb *Mac OS X* usant la llibreria *Pcap4j*. Les classes que s'han modificat són totes les relacionades amb la captura de paquets com *Capturer* o *Packet Capturer* però també les que els gestionaven com *Flow Generator* o *SocketInforGenerator*.

A més a més, també s'han solucionat molts errors relacionats amb altres dependències que no linkaven correctament. Les llibreries de *Sigar*, *log4j* i *SQLite*, d'entre altres, s'han hagut de descarregar de nou amb el format correcte i compilar-les per incloure-les en el projecte. També s'han arreglat problemes d'incompatibilitat d'altres classes o scripts com el

wrapper.sh, la eina *globalIP*, les classes que cridaven a *MySQL*, els *daemons* o classes del *wrapper* que s'han hagut de modificar i compilar de nou usant el programa *gradle*.

Per últim, un cop s'ha aconseguit que funcionés correctament el software en *Mac OS X*, s'ha realitzat una petita comparativa entre dues *eines DPI*, *nDPI* i *Libprotoident*, per observar quin comportament tenien davant de tràfic provinent d'aquest sistema operatiu. El tràfic, s'ha generat des d'un portàtil *MacBook Air* usant aplicacions úniques del sistema operatiu com *Spotlight*, *iMessages*, *Safari* o *AppStore* i també usant *Google Chrome*, *Dropbox* i *Telegram* per a poder ampliar la comparació.

Els resultats no són significatius a causa de que la mostra és molt reduïda i a més a més, hi ha aplicacions en *Mac OS X* que tot i ser *sockets* diferents, tenen el mateix nom i això provoca que sigui més complicat distingir els paquets. No obstant, a primera vista sembla que *nDPI* ha sigut capaç de separar millor les traces depenent de l'aplicació que no pas *Libprotoident*.

Per tant, el *software VBS* funciona en sistemes *Windows*, *Linux* i *Mac OS X* i està preparat per a realitzar un estudi a gran escala de totes les *eines DPI* que hi ha al mercat. Això aportaria una actualització de l'estudi realitzat el 2014 ja que es podria incloure tràfic i aplicacions provinents d'ordinadors amb *Mac OS X* i noves versions dels sistemes operatius. A més a més, també es podrien comparar els resultats de les *eines DPI* de l'estudi de fa dos anys per saber si hi ha hagut millores.

Bibliografia

[1] T. Riaz J. Pedersen T. Bujlow, K. Balachandran. Volunteer-based system for classification of traffic in computer networks. 2011.

[2] Wikipedia. Eines dpi. https://en.wikipedia.org/wiki/Deep_packet_inspection.

[3] Pere Barlet-Ros Tomasz Bujlow, Valentín Carela-Español. Extended independent comparison of popular deep packet inspection (dpi) tools for traffic classification. 2014.

[4] HowStuffWorks. Estructura d'un paquet. http://computer.howstuffworks.com/question5251.htm.

[5] Am I unique. https://amiunique.org/stats/.

[6] Scrum.

[7] Oracle. Mysql. http://www.mysql.com/.

[8] Wikipedia. Pcap. https://en.wikipedia.org/wiki/Pcap.

[9] Dan Kennedy D. Richard Hipp and Joe Mistachkin. Sqlite. https://sqlite.org/, 2005.

[10] Apache Software Foundation. log4j. https://support.hyperic.com/display/SIGAR/Home, 1999.

[11] Hyperic HQ. Sigar. http://logging.apache.org/log4j/2.x/. [Github] https://github.com/hyperic/sigar.

[12] Wikipedia. Cpu. https://en.wikipedia.org/wiki/Central_processing_unit.

[13] Wikipedia. Ram. https://en.wikipedia.org/wiki/Random-access_memory.

[14] Sly Technologies Inc. jnetpcap. http://jnetpcap.com/.

[15] Wikipedia. Wrapper. https://en.wikipedia.org/wiki/Service_wrapper.

[16] Luis MartinGarcia. libpcap. http://www.tcpdump.org/, 2010.

[17] Winpcap. https://www.winpcap.org/.

[18] Oracle. Java sdk. http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html.

[19] INC SAVVIUS. Wildpackets. http://www.wildpackets.com/resources/compendium/ethernet/ethernet_packets.

[20] Wikipedia. Ipv4. https://en.wikipedia.org/wiki/IPv4.

[21] Wikipedia. Ipv6. https://en.wikipedia.org/wiki/IPv6.

[22] Wikipedia. Icmp. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[23] Wikipedia. Http. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[24] Wikipedia. Tcp. https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[25] Wikipedia. Udp. https://en.wikipedia.org/wiki/User_Datagram_Protocol.

[26] Kaito Yamada. Pcap4j. https://github.com/kaitoy/pcap4j, 2011.

[27] Wikipedia. Jni. https://en.wikipedia.org/wiki/Java_Native_Interface.

[28] Apache Software Fundation. Yajsw. http://yajsw.sourceforge.net/.

[29] Wikipedia. Netstat. https://en.wikipedia.org/wiki/Netstat.

[30] Microsoft. https://technet.microsoft.com/en-us/sysinternals/tcpview.aspx, 2011.

[31] Linux. losf. https://linux.die.net/man/8/lsof.

[32] GitHub. Opendpi. https://github.com/thomasbhatia/OpenDPI.

[33] GitHub. ndpi. https://github.com/ntop/nDPI.

[34] GitHub. Libprotoident. https://github.com/wanduow/libprotoident.

[35] GitHub. Libprotoident reader. https://github.com/lmangani/libprotoidentReader.

[36] Tanuki Software. Error wrapper. https://wrapper.tanukisoftware.com/doc/english/prop-startup-timeout.html.

[37] SuperUser. Error launchctl. http://superuser.com/questions/793872/can-t-launch-daemon-with-launchctl-in-yosemite.

[38] StackOverflow. Error launchctl. http://stackoverflow.com/questions/31001857/osx-why-my-launchagent-process-starting-as-root-instead-or-current-user.

[39] ServerFault. Error launchctl. http://serverfault.com/questions/194832/how-to-start-stop-restart-launchd-services-from-the-command-line.

[40] Gradle. Gradle. https://gradle.org/.

[41] StackOverflow. Error gradle. http://stackoverflow.com/questions/22307516/gradle-finds-wrong-java-home-even-though-its-correctly-set.

[42] Coderanch. Error gradle. http://www.coderanch.com/t/582446/java/java/access-sun-management-OperatingSystemMXBean-eclipse.

[43] Oracle. Error gradle. https://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/management/package-summary.html.

[44] StackOverflow. Error sqlite. http://stackoverflow.com/questions/15559551/unsatisfiedlinkerror-with-sqlite4java-jar-on-mac-os-x-netbeans.

[45] Google Code. Llibreria sqlite. https://code.google.com/archive/p/sqlite4java/issues/56.

[46] StackOverflow. Error dataextracter servidor. http://stackoverflow.com/questions/35186/how-do-i-fix-a-nosuchmethoderror.

[47] jNetPcap. Unsatisfiedlinkerror. http://jnetpcap.com/?q=node/1245.

[48] StackOverflow. Path jnetpcap. http://stackoverflow.com/questions/5419039/is-djava-library-path-equivalent-to-system-setpropertyjava-library-path.

[49] Chilkatsoft. Carregar llibreries mac os x. https://www.chilkatsoft.com/java-loadLibrary-MacOSX.asp.

[50] Apple Developer. Flags. https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/DynamicLibraryUsageGuidelines.html.

[51] Apple Developer. Definició funció dlopen. https://developer.apple.com/legacy/library/documentation/Darwin/Reference/ManPages/man3/dlopen.3.html.

[52] Unity3D. Error llibreria dlopen. http://answers.unity3d.com/questions/1114671/how-to-use-dlopen-on-os-x.html.

[53] StackOverflow. Conversió .so a .jnilib. http://stackoverflow.com/questions/1495161/converting-a-so-file-to-a-jnilib-file/1495235#1495235.

[54] StackOverflow. Diferències .so a .dylib. http://stackoverflow.com/questions/2339679/what-are-the-differences-between-so-and-dylib-on-osx.

[55] Apple Developer. Llibreries dinàmiques. https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/OverviewOfDynamicLibraries.html#//apple_ref/doc/uid/TP40001873-SW1.

[56] Chuidiang. Llibreries estàtiques. http://www.chuidiang.com/clinux/herramientas/librerias.php.

[57] Markmail. Diferències jnlib i dylib. http://markmail.org/message/cksb24oiwjszohvl.

[58] Hohonuuli. Jni. http://hohonuuli.blogspot.com.es/2013/08/a-simple-java-native-interface-jni.html.

[59] Chilktsoft. Crear dylib. https://www.chilkatsoft.com/java-loadLibrary-MacOSX.asp.

[60] Adamish. Test jni. http://adamish.com/blog/archives/620.

[61] StackOverflow. Instal·lació ant. http://stackoverflow.com/questions/3222804/how-can-i-install-apache-ant-on-mac-os-x.

[62] StackOverflow. Error cpptasks. http://stackoverflow.com/questions/5638149/compiling-a-c-program-using-ant-contrib-using-cpptasks.

[63] Sourceforge. Instal·lació cpptasks. http://ant-contrib.sourceforge.net/cpptasks/index.html.

[64] StackOverflow. Path ant. http://stackoverflow.com/questions/29018714/how-to-set-default-ant-path-in-mac-os-x.

[65] StackOverflow. Struct packet_state_t. http://stackoverflow.com/questions/26880048/error-field-has-incomplete-type.

[66] StackOverflow. Arxius .o. http://stackoverflow.com/questions/14201710/create-a-single-o-file.

[67] Github. Instal·lació jna. https://github.com/java-native-access/jna.

[68] slf4j. Llibreria slf4j. http://www.slf4j.org/.

[69] Github. Instal·lació libpcap. https://github.com/Homebrew/legacy-homebrew/issues/34063.

[70] Github. Pcapnetworkinterface. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/core/PcapNetworkInterface.html.

[71] Github. Pcaphandle. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/core/PcapHandle.html.

[72] Github. Packetlistener. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/core/PacketListener.html.

[73] Oracle. Executorservice. https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html.

[74] Oracle. Task. https://docs.oracle.com/javafx/2/api/javafx/concurrent/Task.html.

[75] Github. Ipv4header. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/IpV4Packet.IpV4Header.html.

[76] Gasmi. Convertidor payload. https://www.gasmi.net/hpd/.

[77] Oracle. Inet4address. http://docs.oracle.com/javase/6/docs/api/java/net/Inet4Address.html?is-external=true.

[78] Github. Ipnumber. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/namednumber/IpNumber.html.

[79] Github. Tcppacket. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/TcpPacket.html.

[80] Github. Udppacket. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/UdpPacket.html.

[81] Github. Arppacket. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/ArpPacket.html.

[82] Github. Ethernetpacket. https://kaitoy.github.io/pcap4j/javadoc/latest/en/org/pcap4j/packet/EthernetPacket.html.

[83] StackExchange Security. Ssl imessages. http://security.stackexchange.com/questions/18908/the-inner-workings-of-imessage-security.

[84] NakedSecurity Sophos. Ssl appstore. https://nakedsecurity.sophos.com/2013/03/09/apple-finally-adopts-https-for-the-app-store-here-is-why-it-matters/.

[85] Wikipedia. Ssl/tls. https://en.wikipedia.org/wiki/Transport_Layer_Security.

A. Comandes

En aquest apartat es detallen totes les comandes usades durant el desenvolupament del projecte. S'ha de tenir en comtpe que s'ha prescindit de l'ús d'IDEs, per tant totes les comandes estan pensades perquè s'executin a través de terminal.

Arxius JAR

Arxius importants

Els següents dos arxius jar s'han d'anar actualitzant cada vegada que es modifica algun .java de la carpeta *VBSeclipsePayloads/*:

- VBSclient_64bit/app/client.jar
- VBSserver/app/server.jar

Comandes

Crear JAR:

```
jar -cf fitxer.jar fitxers.class
```

Actualitzar JAR:

```
jar -uf fitxer.jar fitxers.class
```

Veure el contingut del JAR:

```
jar -tf fitxer.jar
```

Eliminar un .class del JAR:

```
zip -d fitxer.jar ruta_fitxer/fitxer.class
```

Executar un .class determinat del JAR

```
jar -cp fitxer.jar ruta_fitxer/fitxer.class
```

Gradle

Si es modifica algun arxiu Java de *YAJSW*, s'ha de compilar de la següent manera:

1. Anar a la carpeta del Gradle: VBSclient_64bit/build/gradle/.
2. Executar la comanda:

   ```
   sudo -E ./gradlew.sh -s
   ```

3. Copiar l'arxiu wrapper.jar de VBSclient_64bit/build/gradle/wrapper/build/libs/ a VBSclient_64bit/
4. Copiar l'arxiu wrapperApp.jar de VBSclient_64bit/build/gradle/wrapper-app/build/libs/ a VBSclient_64bit/

Si es fa alguna modificació en el servidor, s'han de seguir els mateixos passos però en la carpeta *VBSserver*.

Compilar classes VBS

Sempre que es modifiqui algun arxiu Java de la carpeta *VBSeclipsePayloads/src/* s'han de compilar indicant totes les llibreries i s'han d'afegir els .class resultants al jar corresponent, client.jar o server.jar:

1. Compilar usant la següent comanda (és molt important indicar totes les llibreries que s'usen:

   ```
   javac -cp
   .:../lib/log4j-1.2.16.jar:
   ../lib/sqlite4java.jar:
   ../sigar/sigar.jar:
   ../lib/jnetpcap.jar:
   ../lib/pcap4j-core-1.6.7-SNAPSHOT.jar:
   ../lib/pcap4j-packetfactory-propertiesbased-1.6.7-SNAPSHOT.jar:
   ../lib/pcap4j-packetfactory-static-1.6.7-SNAPSHOT.jar:
   ../lib/pcap4j-packettest-1.6.7-SNAPSHOT.jar:
   ../lib/pcap4j-sample-1.6.7-SNAPSHOT.jar:
   ../lib/jna-4.2.2.jar:../lib/slf4j-api-1.7.21.jar:
   ```

```
../lib/slf4j-nop-1.7.21.jar
vbsClient/*.java packetCapturer/*.java socketMonitor/*.java
datastructure/*.java databaseHandler/*.java flowGenerator/*java
dataTransmitter/*.java
```

2. Amb els .class que s'han generat de la compilació, s'han d'afegir al .jar corresponent:

```
jar -uf ../../VBSclient_64bit/app/client.jar packetCapturer/*.class
vbsClient/*.class socketMonitor/*.class datastructure/*.class
databaseHandler/*.class flowGenerator/*.class dataTransmitter/*.class
```

Cal tenir en compte que els paths poden variar depenent d'on estiguin les diverses carpetes. Si es fa alguna modificació en el servidor, s'han de seguir els mateixos passos però en la carpeta *VBSserver*.

Compilar jNetPcap

Per compilar la llibreria s'ha de fer amb la comanda javac executant-se des de la carpeta més externa. És a dir, des del nivell superior a la carpeta */org* o */com*. També s'han d'incloure les llibreries necessàries:

```
javac -cp .:../lib/log4j-1.2.16.jar
com/slytechs/library/*.java org/jnetpcap/*.java
org/jnetpcap/nio/*.java org/jnetpcap/packet/*.java
org/jnetpcap/protocol/*.java org/jnetpcap/util/*.java
```

Per últim, s'ha d'afegir a *client.jar*:

```
jar -uf ../../VBSclient_64bit/app/client.jar com/slytechs/library/*.class
org/jnetpcap/*.class org/jnetpcap/nio/*.class org/jnetpcap/packet/*.class
org/jnetpcap/protocol/*.class org/jnetpcap/util/*.class
```

Podria ser que alguna versió més nova del jNetPcap o si s'usa un IDE en comptes d'executar-ho per consola, es pugui compilar utilitzant algun script intermig.

És molt important que quan s'executi la comanda del jar, es faci des de la carpeta del nivell superior a */org* o */com*.

Compilar Pcap4j

Per compilar la llibreria només cal executar la comanda:

```
sudo mvn install
```

El programa ja té totes les dependències i crearà tots els .class i jar necessàris. Només caldrà que afegim les classes a *client.jar*:

```
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/core/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/packet/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/packet/constant/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/packet/factory/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/packet/namednumber/*.class
jar -uf VBSclient_64bit/app/client.jar org/pcap4j/util/*.class
jar -uf VBSclient_64bit/app/client.jar org/slf4j/*.class
jar -uf VBSclient_64bit/app/client.jar org/slf4j/event/*.class
jar -uf VBSclient_64bit/app/client.jar org/slf4j/spi/*.class
jar -uf VBSclient_64bit/app/client.jar org/slf4j/helpers/*.class
jar -uf VBSclient_64bit/app/client.jar com/sun/jna/*.class
jar -uf VBSclient_64bit/app/client.jar com/sun/jna/ptr/*.class
jar -uf VBSclient_64bit/app/client.jar org/slf4j/impl/*.class
```

Igual que els apartats anteriors, la comanda jar s'ha d'executar des de la carpeta del nivell superior a */org* o */com*.

PATHS

En aquesta secció es detalla com es comporta el sistema Mac OS X a l'hora de buscar llibreries dinàmiques i quines són les variables globals que usa.

Quan el nom de la llibreria és el nom d'un arxiu (no s'especifica cap directori), el sistema operatiu busca la llibreria a diversos llocs fins que la troba, seguint aquest ordre:

1. Variable LD_LIBRARY_PATH
2. Variable DYLD_LIBRARY_PATH
3. Directori des d'on s'ha executat

4. Variable DYLD_FALLBACK_LIBRARY_PATH

Quan el nom de la llibreria conté com a mínim un directori (s'especifica directori), el sistema operatiu busca la llibreria en el següent ordre:

1. Variable DYLD_LIBRARY_PATH buscant pel nom del fitxer
2. Directori que s'ha indicat
3. Variable DYLD_FALLBACK_LIBRARY_PATH buscant pel nom del fitxer

Aclaracions

1. Si no s'especifica el DYLD_FALLBACK_LIBRARY_PATH, dlopen suposarà que el valor de la variable és:

   `$HOME/lib:/usr/local/lib:/usr/lib`

2. Si no s'especifica el DYLD_FALLBACK_FRAMEWORK_PATH, dlopen suposarà que el valor de la variable és:

   `$HOME/Library/Frameworks:/Library/Frameworks:/Network/Library/Frameworks:/System/Library/Frameworks`

3. Mac OS X utilitza fitxers "universals" per combinar llibreries de 32 bits i 64 bits. Això implica que no hi ha directoris que separin els dos tipus.
4. Si cap de les variables RTLD_LAZY o RTLD_NOW s'especifica, per defecte s'agafarà la RTLD_LAZY.
5. Si cap de les variables RTLD_GLOBAL o RTLD_LOCAL s'especifica, per defecte s'agafarà la RTLD_GLOBAL.

JNI

En aquest apartat s'explica com generar un .dylib des d'un arxiu Java i des d'un arxiu de C++. Per fer-ho més senzill es amb un exemple d'un HelloWorld.

Des de Java

Primer de tot es compila el .java per obtenir el .class:

```
javac HelloWorld.java
```

A continuació es genera el .h (headers en JNI) usant els .class. Si s'usessin llibreries externes, s'haurien d'indicar amb el flag -classpath:

```
javah -jni -classpath . HelloWorld
```

Es crea un arxiu .c o cpp on es programa les funcions. Un cop fet es compila i s'obte el dylib:

```
gcc -shared libHelloWorld.cpp
-I-I/System/Library/Frameworks/JavaVM.framework/Headers
-I/System/Library/Frameworks/JavaVM.framework/Headers
-I/System/Library/Frameworks/JavaVM.framework/Headers/darwin/
-o libhelloworld.dylib
```

Per últim, s'ha de copiar la llibreria a la carpeta */usr/local/lib*.

Des de C++

Tenint els fitxer .cpp i .h amb les funcions i headers corresponents, s'executa la següent comanda per generar el .dylib:

```
g++ -dynamiclib -o libHelloWorld.dylib libHelloWorld.cpp
```

Quan es compili algun codi que usi aquesta llibreria, s'ha d'afegir el flag "-l(nom de la llibreria". Per exemple, si la llibreria es diu *libHelloWorld*, el flag serà *-lHelloWorld*.

Per últim, s'ha de copiar la llibreria a la carpeta */usr/local/lib*.